Ui Development

UI Development is the process of developing a Front End of any websites using various UI Technologies like HTML, CSS, JavaScript, JQuery, Bootstrap, and AngularJS for better interaction with a user and the underlying system.

Prepared by,

MR. NAVEEN SAGGAM

(Oracle Certified Professional)

Website : uiBrains.com
GitHub : https://github.com/thenaveensaggam

Ui Development

Definition

UI Development is the process of developing a Front End of any websites using various UI Technologies like HTML, CSS, JavaScript, JQuery, Bootstrap, and Angular JS for better interaction with a user and the underlying system.

Introduction to UI

There are different names for UI like GUI, UI, UX and front End.

```
GUI -> Graphical User Interface
UI -> User Interface
UX -> User Experience
FontEnd -> The Font Look and Feel of any website
```

Initial days of communication between two human beings, it required a language and body language. The same if a person wants to communicate with a machine it requires the concept called.

First we needs to understand why every software industry starts using the concept called UI. For any kind of business or any product to reach the intended users it should have a proper website. The website or a mobile app is the only way to interact with any user.

But here with a mobile app, a user can easily interact with the underlying system, but a user cannot install all the mobile apps in his mobile there may be a memory issue. The only best way is a website and if that website works like a mobile app then that will be more convenient for a user.

To develop a website for any underlying system we should learn UI Technologies. The UI development is the process of developing any website's front end using various UI Technologies. UI Technologies is not a single technology, which is the combination of other and each different technologies.

First we understand a website's architecture.

Each website's architecture contains majorly 3 parts, such as

1) Client
2) Server
3) Database

Let's understand how the Facebook application works as per the above architecture.

A User opens a Facebook login page in his client's browser and clicks on 'login' button. Then that browsers collects the form data and sends a HTTP request to a Facebook server.

The Facebook server performs some processing and checks the user verification with the Facebook database. The Facebook database executes few SQL queries and gets the user data and confirms the user verification with the Facebook server.

Facebook server collects the user data and prepares the complete Facebook page and sends back to client's browser, so that the user can able to see his Facebook page with complete Notifications, Chat information and friend requests.

Here we have to understand the purpose of client, server and database is completely different.

To develop the client side UI, server side code and Database design we should require different kinds of technologies.

Technologies required for web development

Client Side	Server side	Data Bases
HTML	.NET	Oracle database
CSS	Java	MySQL Database
JavaScript	PHP	DB2 from IBM
JQuery , AJAX	Node JS , Express JS	SQL Server
Bootstrap	SAP	Mongo DB
Angular / Angular JS	Sales Force	Firebase

As a UI Developer we are responsible to develop the complete front end of any website. For this we are learning the Client side technologies / UI Technologies.

Let's understand the brief introduction about the UI technologies.

HTML

HTML stands for Hypertext markup language which is used to structure a webpage. We use HTML just to define the structure of a webpage with some basic styles.

CSS

CSS stands for Cascading Style Sheet. The CSS is used to present the data on the webpage. CSS is used to override the default styles of HTML and apply various styles to a webpage.

JavaScript

JavaScript is a Client side programming language which is used to display dynamic data on a webpage using the concept called DOM Manipulation.

JQuery

JQuery is a library of JavaScript, which is also used to display dynamic data on a webpage in a simplified manner and DOM Manipulation and CSS Manipulation AJAX calls in a simplified manner.

AJAX

AJAX stands for Asynchronous JavaScript and XML, this is a client side programming technique to update the specific parts of a webpage without reloading the whole browser. Here we use a built in browser's object called XMLHttpRequest Object.

Bootstrap

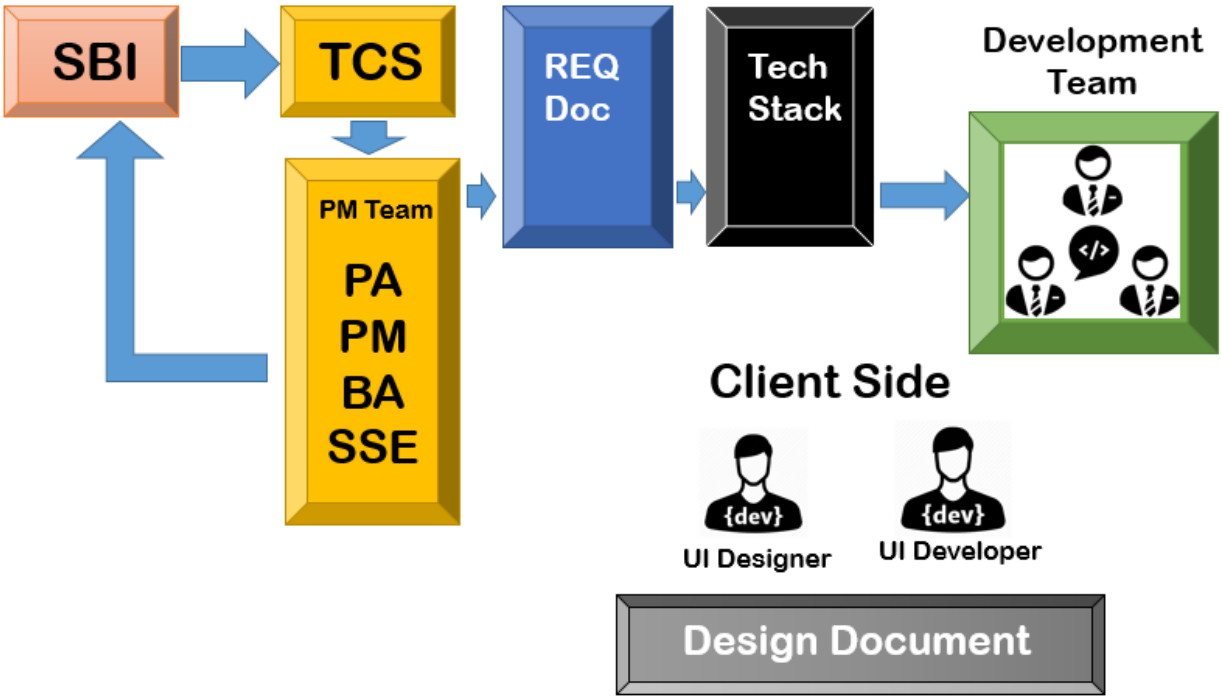
This is a Framework of HTML, CSS, and JavaScript. The Core features of Bootstrap is Responsiveness and Mobile first approach. This is the world's most popular front end framework for responsive web applications development any website developed by using Bootstrap, which works properly in all the browsers , all the devices and all the operating systems.

Angular JS

This is most popular JavaScript framework developed and maintained by Google. We use Angular JS to develop rich client side applications with complex logics. We can simplify the complex front end logics by using MVC architecture and two-way data binding. This Angular JS framework is best suitable for Single page applications.

Understanding Web Development

UiBrains.com



Let's understand how a software company gets the projects and how they starts the development of the projects.

A Client Organization required a website for their business. For this they approaches a software company. Client Organization wants to create a website, but they don't have proper idea of the website can be built for their requirement.

Software Organization forms a project management team which contains project architect, Project Manager, Business Analysts and few senior software engineers. The Software Organization sends the Project Management team to the main branch of Client organization for the complete study of the requirement.

Once the complete analysis is done, the project management team prepares a document called Requirement Document. Once the Requirement document is finalized then The Project Architect decides the technologies to be used to develop the project, he decides the technologies to be used for client side, server side and database side technologies.

Once the Technology Stack is ready, they recruits or forms development team for the project. The Development team contains three types of developers such as

1) Client Side Developers
2) Server Side Developers
3) Database Developers

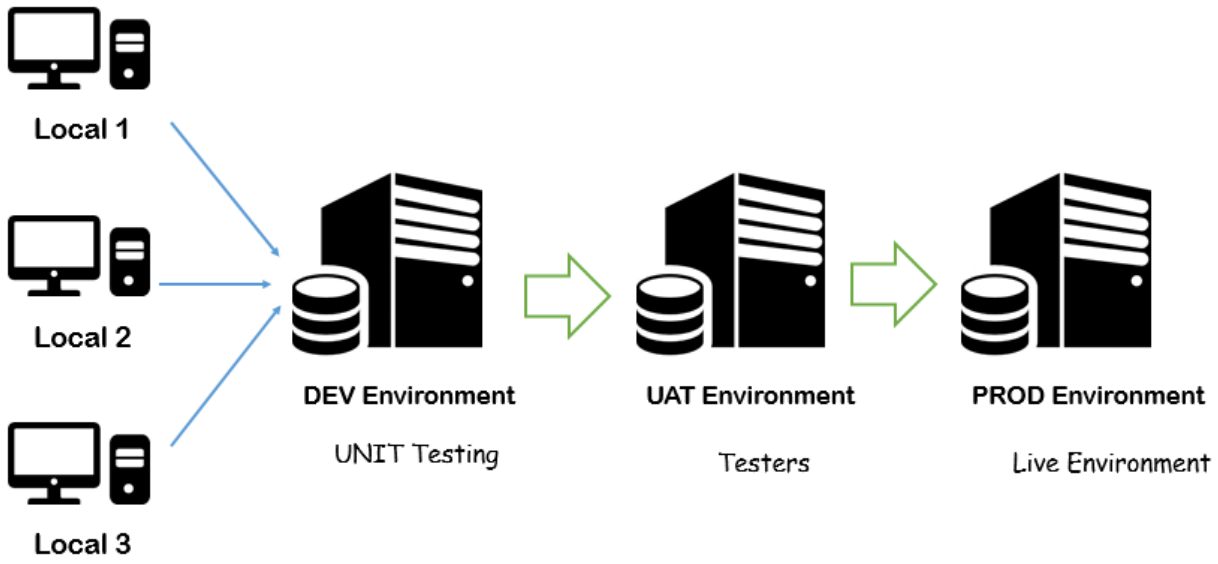
At the client side, there are again two kinds of software Engineers like UI Designers, UI Developers. First the requirement document is shared with UI Designer.

UI Designer understand the complete Requirement document and he proposes two or three designs using his creative skills and with UI Design knowledge with the required UI Design tools. Once the designs are ready, Client selects a design for the project.

Once the design is ready, UI Designer prepares a design Document and shares with UI developer. Now UI developer gets the UI Design document and starts the development of the complete front end of the project.

Environments for any Project Development

There are various stages or environments for every projects development as follows,



For every project development there may be three or four developers for UI. Each developer has to develop their code in the local environment. The company provided Desktop or laptop where the development project starts is called as Local environment.

Before starting any project, all the developers has to install the required technologies setup or software in their local environment. This process is called as development environment setup.

Once the setup is ready then all the developers start development of the project in their local environments. Once the development is ready in their local environments they will integrate the code in DEV Environment.

Once the code is integrated with the server side and database code, all the developers will perform one round of testing this is called as Unit Testing.

Once Unit testing is finalized then they will push the code to UAT environment.

UAT stands for User Acceptance Test, this is the environment is accessible for all the testers of the project. In this environment all the testers will apply few testing techniques like integration testing and smoke testing.

Once the testing is finalized then the code is ready to push to PROD Environment. This environment is the last stage of any project development, this is also called as LIVE Environment. This environment is accessible to all the customers of the client organization.

Types of Software Companies

The software companies are divided into 3 categories majorly, they are as follows,

Services Based Software Company

Any software company which develop software or websites for a particular Client is called as Service Based software companies.

A Website which is developed for a particular client is applicable for that client only, which will not works for any other clients.

A person wants to work under Services Based software company they should require hands on experience on the respective technologies.

Ex: TCS, IBM, HCL, and Wipro

Product Based Software Company

Any software company which develops software or product not for a particular client, it's for the Universal clients is called as Product Based Software Companies.

These software will be developed to be useful for all the users around the world.

A Person wants to work under Product Based Software Company he should be the Master of the respective technologies.

Ex: Google (Google Drive, Google Maps, Gmail, YouTube, Google Duo, Google Tez)

Microsoft (MS Office, Operating System, .Net language, Visual Studio Editor, Azure Cloud)

Oracle (Oracle Database, Java Language, Weblogic Server)

Note: There are few software companies comes under product based as well as Product based like Oracle, Wipro, HCL, IBM etc.

Solution Based Software Company

Any Software companies which researches the current market trends and analyses the currents Market needs and they come up with an innovative solutions with their products / Websites is called as Solution based Software Company.

A person wants to work under Solution based Software Company, He should be able to create new language or Technology or Tool. Usually Each Solution based companies uses their own software tools.

Ex: Amazon, Facebook, Uber, Redbus, Swiggy etc.

UiBrains.com

Roles of UI Designer

UI Designer interacts with the project management team as well as Business Analysts.

He understands the complete Requirement document and he proposes an innovative solutions with for better user Experience with his UI Design tools.

Skill Set

Complete understanding of project's work flow

Good Presentation skills

Hands on experience on Photoshop

Innovative & Creative Ideas

Basic knowledge of HTML & CSS.

Roles of UI Developer

UI Developer interacts with Project Management Team and UI Designer.

He understands the complete projects workflow as well as UI Design

He is responsible to develop the complete frontend of the project.

Skill Set

Good Programming Skills

Hands on experience on the technical skills of UI

Analytical skills applied with technologies.

Complete understanding of the project workflow.

UiBrains.com

Tools used to the web Development

Software Name	Produced By	Free / Licenced

Notepad	Microsoft	Free
Notepad++	Notepad plus plus	Free
Atom Editor	GitHub	Free
Brackets Editor	Adobe Systems	Free
Visual Studio Code	Microsoft	Free
Sublime Text	Sublime	Semi Licensed
Webstorm	Jet Brains	Fully Licensed
Dream Weaver	Adobe Systems	Fully Licensed

Development Environment Setup

In this setup we will install the required software and editors in our laptop or desktop and proceed with the development of any project.

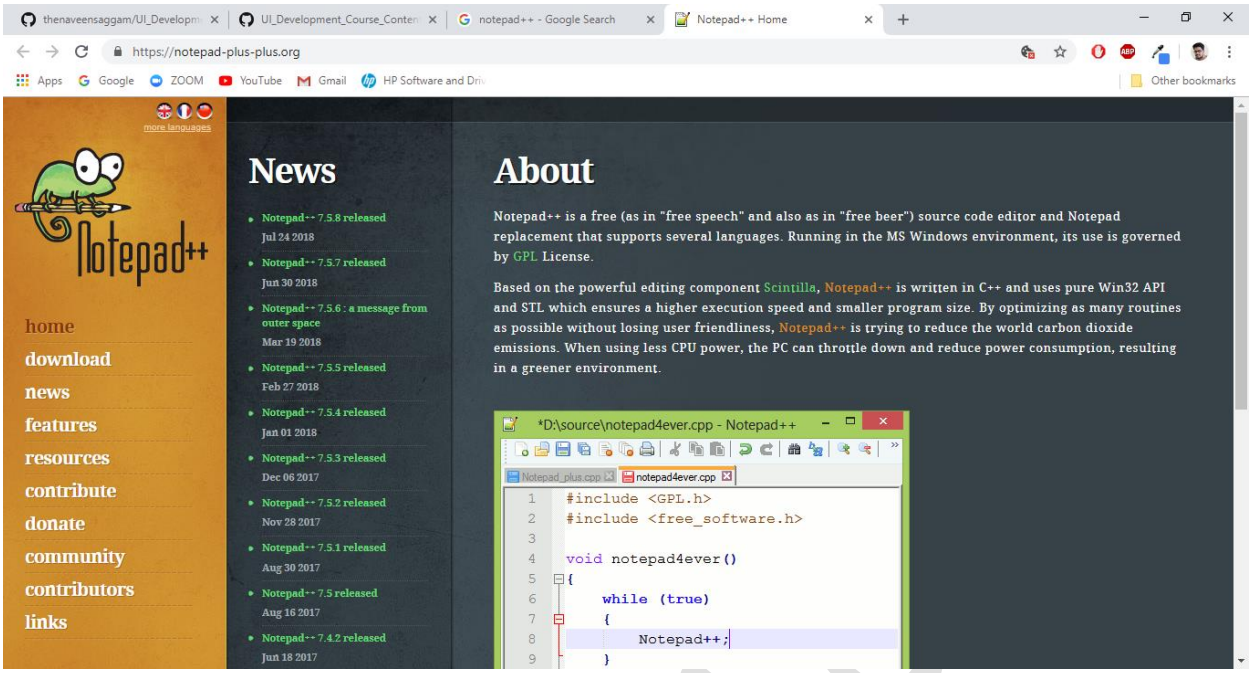
In this setup we are going to install the following softwares,

1) Notepad++
2) Node JS
3) Webstorm
4) Download Source code

Installation of Notepad++

Go to Google and type 'notepad++' or go to official website https://notepad-plus-plus.org/

UiBrains.com

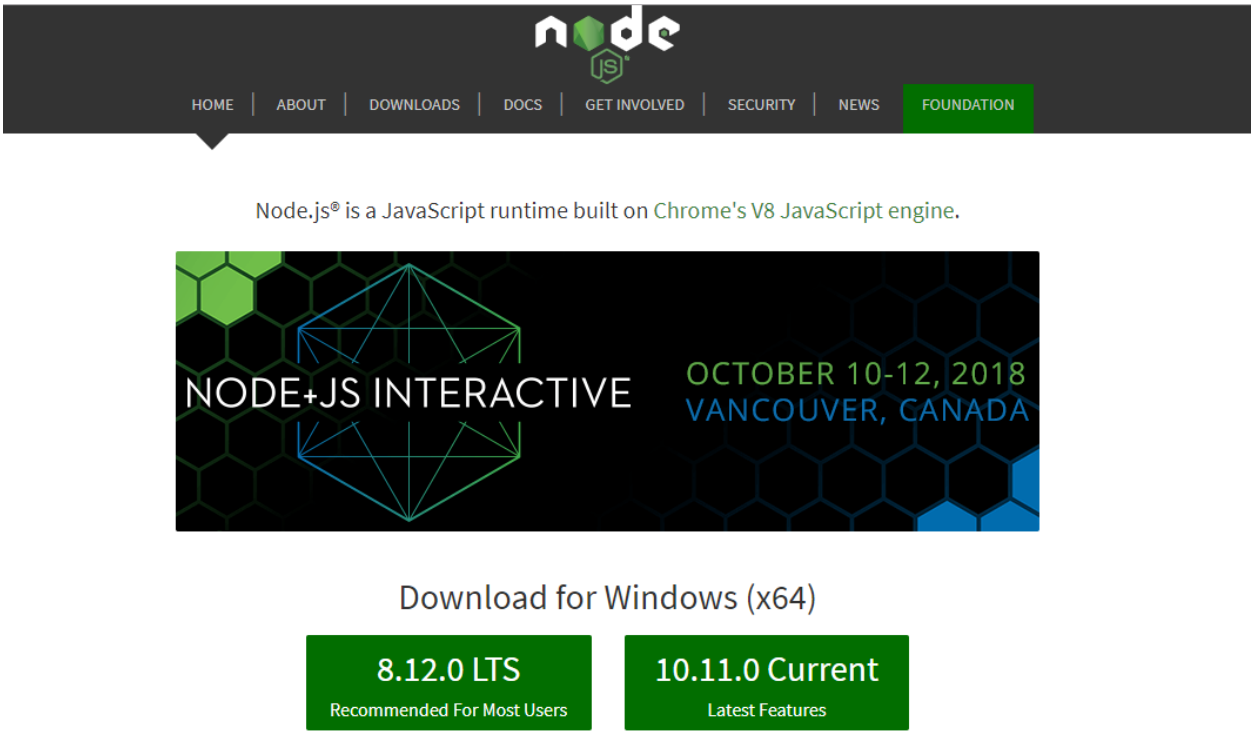


- Go to "Google" -> type "notepad++"
- Go to https://notepad-plus-plus.org/
- click on "download" button
- select the download options like (32 bit / 64 bit)
- Click on the "installer" link
- it downloads a '.exe' file
- double click on the downloaded file
- Proceed with default installation.

Installation of Node JS

Please follow the below steps to install Node JS

- Go to "Google" -> type 'node js"
- Go to https://nodejs.org/en/



-
- click on "8.12.0" Green color button
- it downloads a '.msi' file
- double click on '.msi' file
- Proceed with the default installation.

Verification of Node JS Installation

- Go to "command prompt"
- type 'node -v' to get the version of Node JS

```
C:\Users\NAVEEN SAGGAM>node -v
v8.11.4
```

-
- once you get the version, it is installed successfully
- type 'npm -v' to get version of NPM

```
C:\Users\NAVEEN SAGGAM>npm -v
5.6.0
```

-
- once you get the version, it is installed successfully
- Now you can install 'live server'

Installation of Live Server

- go to "Command Prompt"
- type 'npm install -g live-server" (with internet connection)

```
C:\Users\NAVEEN SAGGAM>npm install -g live-server
```

- Once it is installed, we can check the version of it by using
- 'live-server -v'
- Here we get the version as 'live-server 1.2.0'

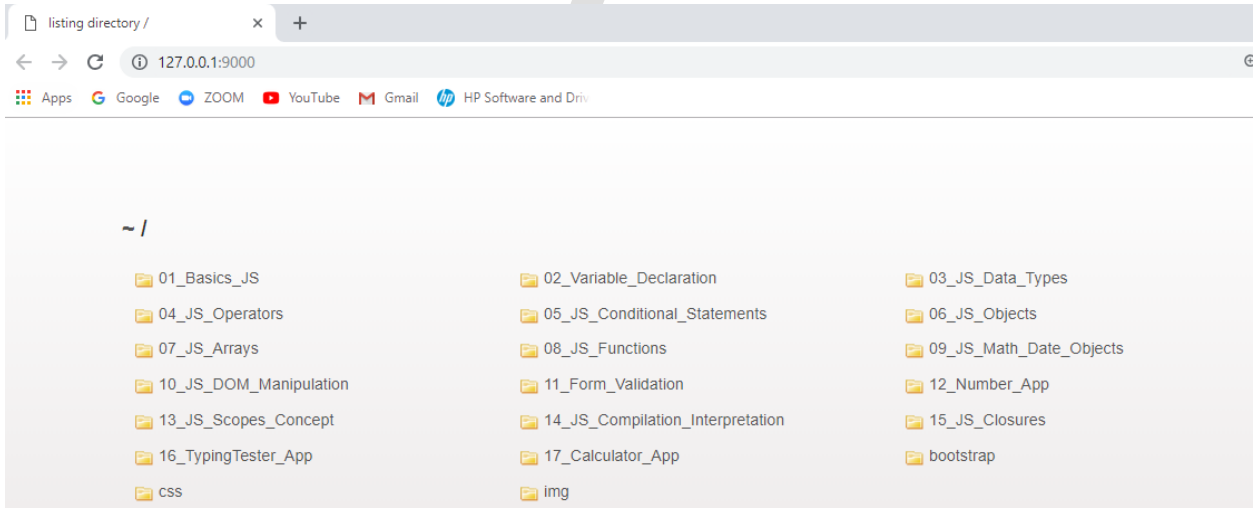
```
C:\Users\NAVEEN SAGGAM>live-server -v
live-server 1.2.0
```

How to start live server

- Go to "Command Prompt"
- Go to "projects folder"
- type 'live-server --port=9000' and click on 'enter'

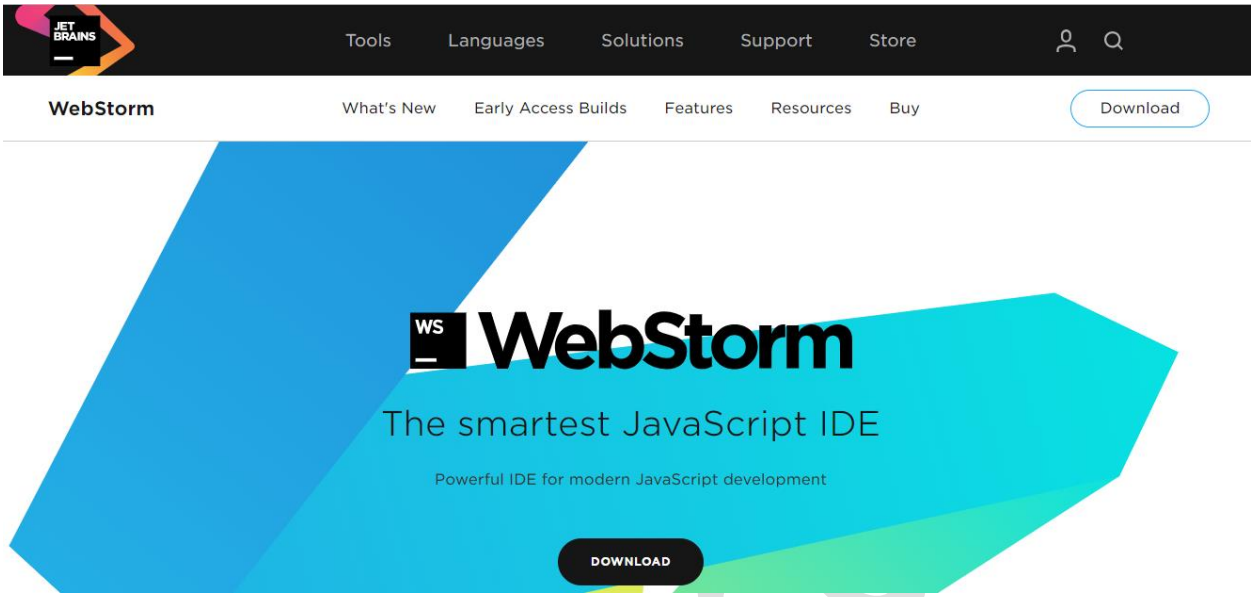
```
D:\Workspace\UI_Developement\Offline_Classes\7AM_BATCH\JavaScript\JS_Completed
_Template>live-server --port=9000
```

- This will start the live server and open the browser automatically.
- With the given address as “127.0.0.1:9000”



How to download Webstorm IDE

- ➔ You can download the Webstorm IDE from the official website as mentioned below.
- ➔ https://www.jetbrains.com/webstorm/

➔ 

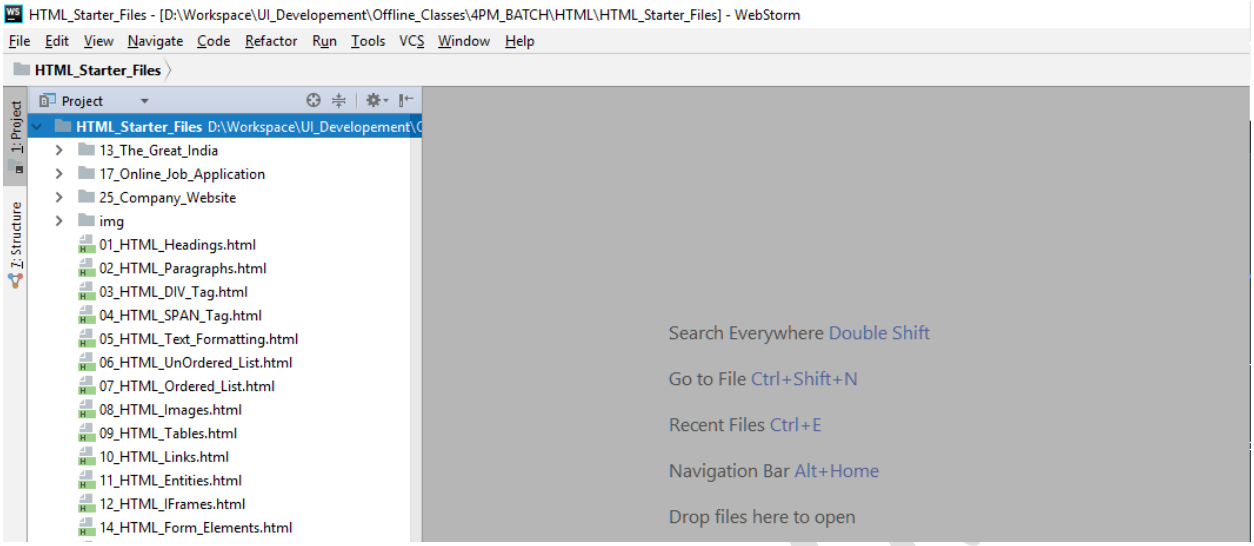
Download Source for the HTML Development

- ➔ go to "google"
- ➔ type "naveen saggam"
- ➔ go to https://github.com/thenaveensaggam
- ➔ click on "HTML_Template"
- ➔ Click on "Clone or Download" green color button
- ➔ it downloads a '.zip' file
- ➔ Extract the zip and create a folder structure

Now let's starts Integrate the above source code with the Webstorm IDE and starts with the project Development.

- ➔ First we have to extract the Zip file of "HTML_Template"
- ➔ Go to "HTML_Starter_Files" Folder
- ➔ Copy the address of this folder
- ➔ Go to Webstorm , click on file -> open -> paste the address of this folder
- ➔ The HTML_Stater_Files will be loaded to the Webstorm.

UiBrains.com



Now let's start with the HTML Learning.

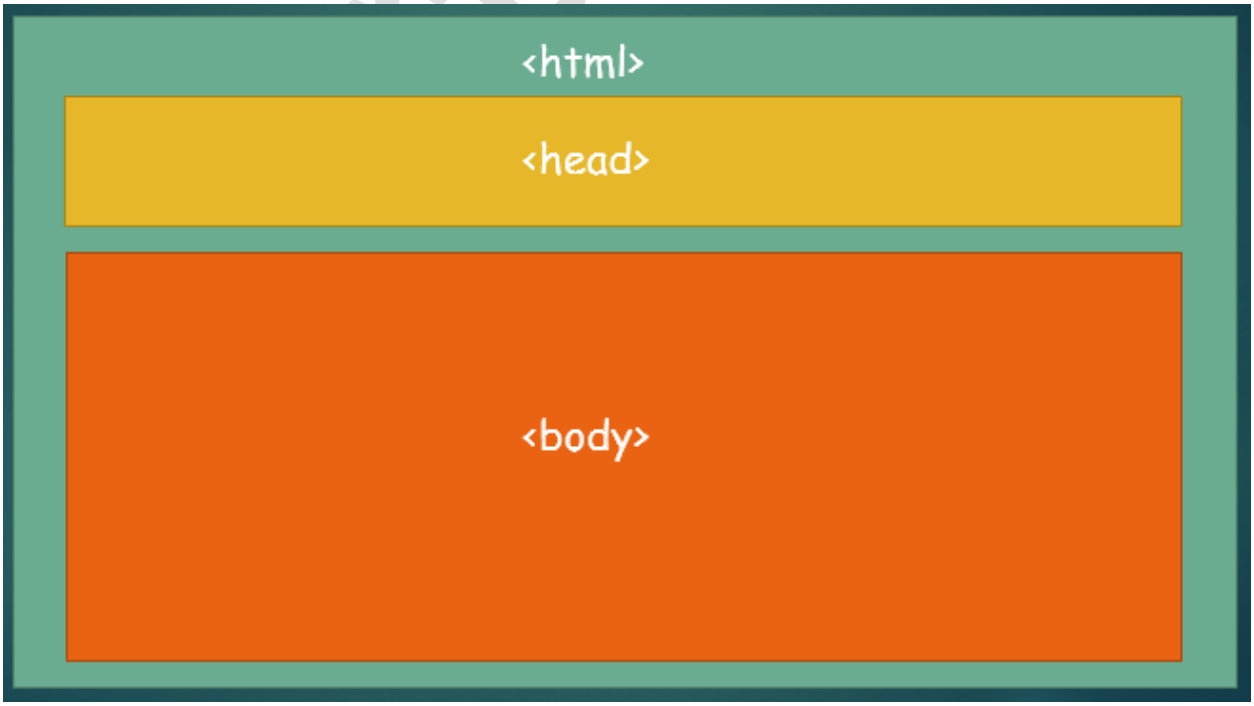
UiBrains.com

HTML

Introduction

HTML Stands for Hypertext markup language, this is one of the core language to develop any website among CSS and JavaScript. We can use the HTML for structuring of a webpage and it is a static programming language.
Any HTML document contains an extension of '.html' or '.htm' which can be run on any browser.
A HTML document contains full of HTML tags and these tags will be parsed by browser and the actual content will be displayed on the browser. Please note that, the HTML tags will not be displayed on the browser and only the content on between the tags will be displayed on the browser.
Here browser is responsible to parse the HTML tags and display content on the webpage or browser.
HTML is a static programming language to display some static content on the webpage. The content can be a paragraph or headings or images without any styles.
The meaning of static is, if any content which is developed by using HTML is always static, it means that content will never change even for multiple reloads of the browser. We can't be able to produce any dynamic content just by using HTML. We can apply the dynamic nature by using the language called JavaScript.

Structure of HTML Document



The complete HTML document is represented by the tag called <html> and it contains two parts as follows,

1) Head Section
2) Body Section

Head Section

The Head Section of a HTML document is represented with the tag called <head>. Any textual content between <head> and </head> comes under head section.

This section contains information like meta information and website's title and few CSS configurations information and few JavaScript Configuration.

Body Section

The Body section of HTML document is represented with the tag called <body>. Any textual content between <body> and </body> comes under body section.

This is one of the important section to display any content on the webpage. Any websites visual area of content will comes under body section. We must write any visual content which we wants to show up on the webpage we must write those tags in body section only.

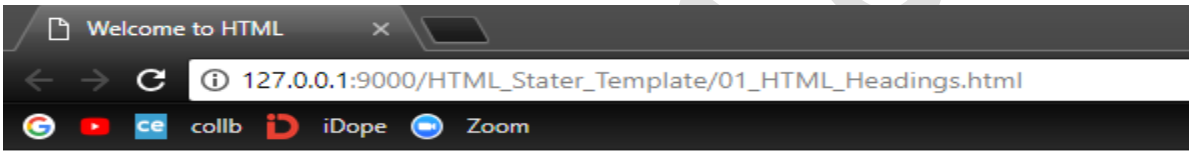
HTML First Application

The basic HTML Document structure looks as follows,

01_HTML_Headings.html

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Welcome to HTML</title>
</head>
<body>

    <h1>Good Morning HTML</h1>

</body>
</html>
```

The above example line number 1 describes the document type. It informs the browser that all the following tags are from HTML and parse and display them as per HTML5 way on the browser.
Line number 2 – 12 indicates the complete HTML Document. And line number 2 is the start tag of HTML and line number 12 is the end tag of HTML.
Line number 3 – 6 indicates the head section of a HTML Document, this contains some Meta information, title, few CSS and JavaScript Configuration.
Line number indicates the Meta information where the character set to be used for a HTML which is the most popular standard given by Unicode Consortium to support almost all languages character set.
Line number 5 indicates, the actual title of the webpage and this title will be displayed in the TAB of the browser of any website.
Line Number 7 – 11 Indicates, the body section of any HTML Document. Any visual content which is displayed on the webpage is from the body section of HTML Document.
Line Number 9 indicates, the main heading of any webpage in HTML Document.
'Good Morning HTML' will be displayed on the webpage. As follows,



Good Morning HTML

Rules for writing HTML Document:

- ➔ Each content which is displaying on the webpage should be written in between a html tag only.
- ➔ All the html tags to be written in lowercase only.
- ➔ Always add an end tag if required, means for some HTML tags / elements are not having a close / end tags. We needs to add an end tag if it is required.

Example:
 , <hr> , <img> tags will not have end tags.
Note: In HTML language HTML tag and HTML Elements are both one and the same. Sometimes we may call it as HTML Tag and sometimes we may call it as HTML Element.

HTML Attributes

The HTML attributes provides an additional information to the HTML Tags, elements.
All the HTML Elements may have attributes but not all the html elements will have same attributes.
All HTML Attributes comes with key and value pairs.
All HTML Attributes should be added to the start tag only.

Example:

```
<input type="text" placeholder="Enter Your Name">
<img src="img/forest.jpg" width="300" height="200">
```

In the above example, the src, width, height are html attributes which adds an extra information to HTML elements.

HTML Header Tags

We can display various headings on the webpage using the HTML Header tags.
In HTML there are 6 types of header tags available such as,

```
1  <h1></h1> -> Main Heading
2  <h2></h2> -> Sub Heading
3  <h3></h3>
4  <h4></h4>
5  <h5></h5>
6  <h6></h6> -> Least Heading
```

Example:

```
1  <h1>Heading 1</h1>
2  <h2>Heading 2</h2>
3  <h3>Heading 3</h3>
4  <h4>Heading 4</h4>
5  <h5>Heading 5</h5>
6  <h6>Heading 6</h6>
```

HTML Heading 1

HTML Heading 2

HTML Heading 3

HTML Heading 4

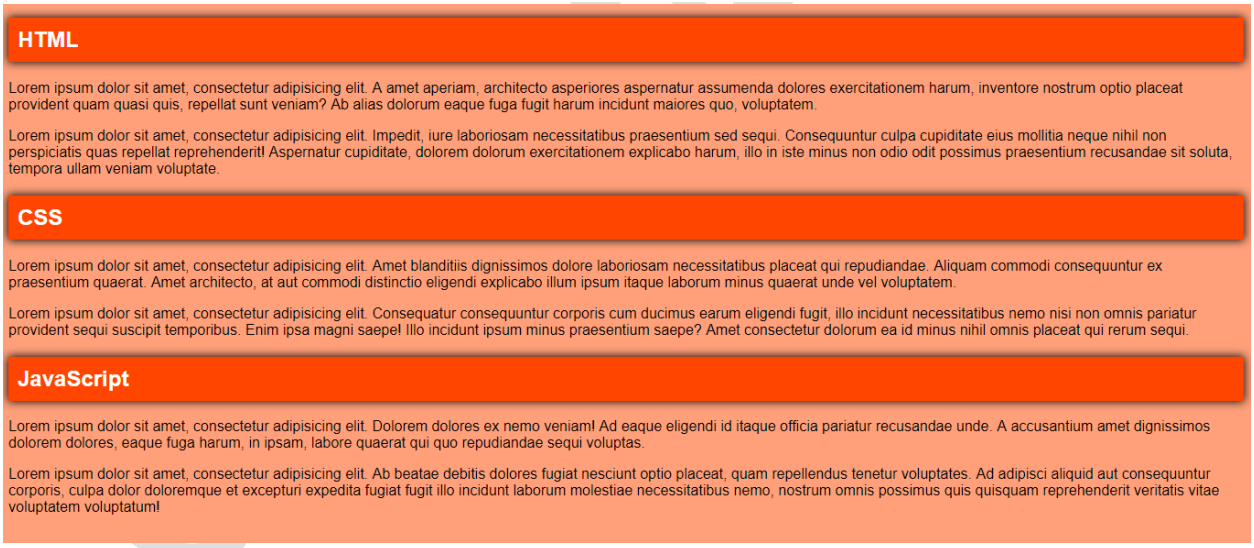
HTML Heading 5

HTML Heading 6

HTML Paragraphs

We can display a paragraph in the webpage or browser using the HTML tag called tag.
Any content to be displayed as a paragraph on the webpage we needs to provide this content between and tags.
Please note that the number of lines which are displayed on the webpage are not same as the number of lines which are written on the webpage.
Browser decides the number of lines to be displayed on the browser based on the width of the browser.
We cannot restricts the number of lines to be displayed on the webpage with the tag.
Note: Normally we never restrict the number of lines for any paragraph.
We can display the same number of lines written in the html document and the number of lines displayed on the webpage using html tag called <pre> tag.
Example:

```
<h2>HTML</h2>
<p>Lorem ipsum dolor sit amet, consectetur adipisicing elit. Accusamus amet animi consequuntur doloribus eaque </p>
<p>Lorem ipsum dolor sit amet, consectetur adipisicing elit. Architecto, commodi deserunt dignissimos</p>

<h2>CSS</h2>
<p>Lorem ipsum dolor sit amet, consectetur adipisicing elit. Adipisci assumenda consectetur debitis</p>
<p>Lorem ipsum dolor sit amet, consectetur adipisicing elit. Alias aliquam</p>
```



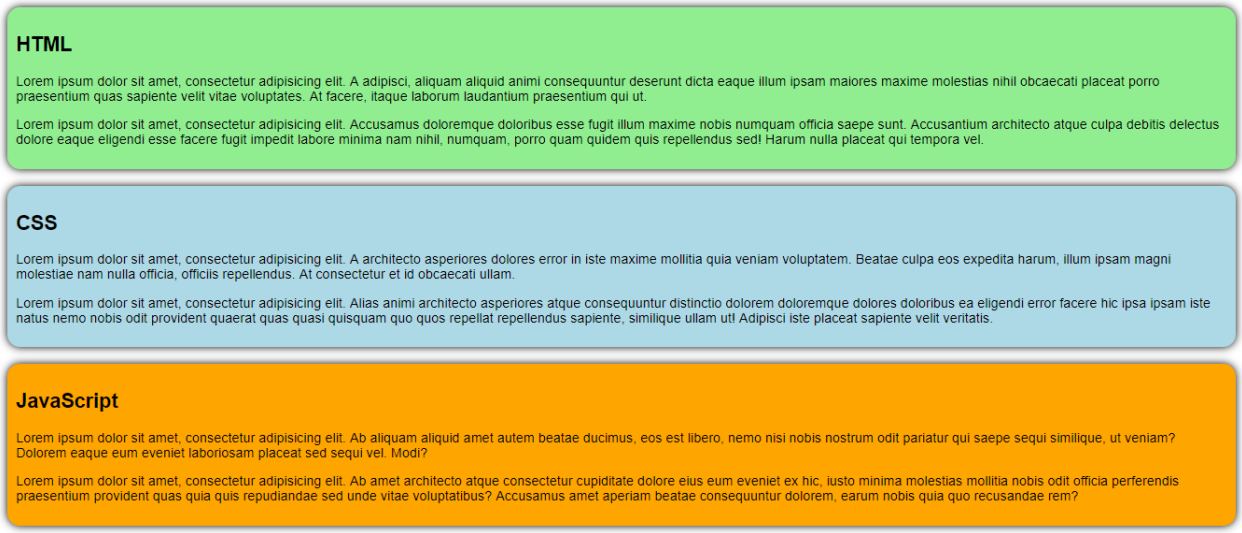
HTML DIV Tag

The HTML DIV tag defines a division or section in an HTML document.
We can divide an html document or the entire webpage into various sections or divisions using DIV tag.
The tag is often used as container for other HTML Elements to style them with CSS.

Within the tags, we can add any number of tags and display like a division or section of content on the webpage. Please note by default no styles will be added for tag. We may have to use CSS to apply custom styles.

Example:

```
<div>
    <h2>HTML</h2>
    <p>Lorem ipsum dolor sit amet, consectetur adipisicing elit.
    deleniti ea eum ex excepturi</p>
    <p>Lorem ipsum dolor sit amet, consectetur adipisicing elit.
    expedita impedit labore natus</p>
</div>

<div>
    <h2>CSS</h2>
    <p>Lorem ipsum dolor sit amet, consectetur adipisicing elit.
    deleniti ea eum ex excepturi</p>
    <p>Lorem ipsum dolor sit amet, consectetur adipisicing elit.
    expedita impedit labore</p>
</div>
```



HTML Span Tag

The tag is an example of inline element. By using tag we can apply some styles for a specific character or a word or a group of words inside a paragraph.

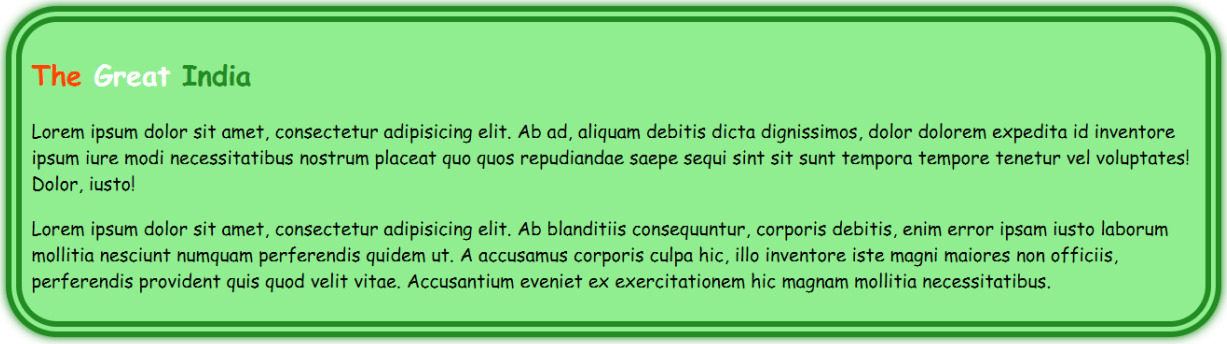
Just by adding a tag, no styles will be applied by default. We can manage the styles by using CSS.

For example we have a content as follows,

```
<p>Lorem ipsum dolor sit amet, consectetur adipisicing elit.
Commodi cupiditate ducimus earum enim error, eveniet iure
laboriosam maiores minus molestiae natus nesciunt, optio,
placeat quia rem saepe sit veritatis voluptate.</p>
```

Here in the above example, If we wants to apply some styles for a specific word called ‘Ipsum’ in the above paragraph, we needs to surround that tag with tag and apply the styles using ‘style’ attribute as follows.

```
<!-- span Tag Usage -->
<div style="...">
    <h2>
        <span style="...">The</span>
        <span style="...">Great</span>
        <span style="...">India</span>
    </h2>
    <p>Lorem ipsum dolor sit amet, consectetur adipisicing elit.
    <p>Lorem ipsum dolor sit amet, consectetur adipisicing elit.
</div>
```



HTML Text Formatting

We can apply various text formatting styles to the HTML content using the following text formatting tags.

```
<b> - for Bold text
<i> - for Italic text
<em> - for Emphasized text
<mark> - for Marked text
<small> - for Small text
<del> - for Deleted text
<ins> - for Inserted text
<sub> - for Subscript text
<sup> - for Superscript text
```

Example:

```
<!-- BOLD TEXT -->
<div>
    <h1>I am <b>BOLD</b> TEXT</h1>
</div>
```

```
<!-- STRONG TEXT -->
<div>
    <h1>I am <strong>STRONG</strong> TEXT</h1>
</div>
```

i am strong text

```
<!-- ITALIC TEXT -->
<div>
    <h1>I am <i>ITALIC</i> TEXT</h1>
</div>
```

i am *italic* text

```
<!-- EM TEXT -->
<div>
    <h1>I am <em>EM</em> TEXT</h1>
</div>
```

i am *em* text

```
<!-- MARK TEXT -->
<div>
    <h1>I am <mark>MARK</mark> TEXT</h1>
</div>
```

i am mark text

```
<!-- SMALL TEXT -->
<div>
    <h1>I am <small>SMALL</small> TEXT</h1>
</div>
```

i am small text

```
<!-- DEL TEXT -->
<div>
    <h1>I am <del>DEL</del> TEXT</h1>
</div>
```

i am ~~del~~ text

```
<!-- INS TEXT -->
<div>
    <h1>I am <ins>INS</ins> TEXT</h1>
</div>
```

i am ins text

```
<!-- SUB TEXT -->
<div>
    <h1>I am <sub>SUB</sub> TEXT</h1>
</div>
```

i am sub text

```
<!-- SUP TEXT -->
<div>
    <h1>I am <sup>SUP</sup> TEXT</h1>
</div>
```

i am sup text

HTML List Tags

We can display any content in the list format using HTML List tags. In HTML there are two types of lists, they are ordered list and Un Ordered List.

UN Ordered List

In HTML the UN ordered list is represented by the tag called <ul> and each list item in un ordered list is represented with the tag called <li>.
By default the list items will be displayed as 'dots', we can change this by using 'type' attribute.
There are four types of UN ordered lists such as disc, circle, square, none.
Example:

```
<ul>
    <li>HTML</li>
    <li>CSS</li>
    <li>JavaScript</li>
    <li>JQuery</li>
    <li>AJAX</li>
    <li>Angular JS</li>
</ul>
```

- HTML
- CSS
- JavaScript
- JQuery
- AJAX
- Bootstrap
- Angular JS

We can change the type of unordered list is by using ‘type’ attribute, we needs to add this attribute to the start tag of <ul> tag only.

Example:

```
<ul type="circle">
     <li>HTML</li>
     <li>CSS</li>
     <li>JavaScript</li>
     <li>JQuery</li>
     <li>AJAX</li>
     <li>Angular JS</li>
</ul>
```

- HTML
- CSS
- JavaScript
- JQuery
- AJAX
- Bootstrap
- Angular JS

Ordered List

In HTML the ordered list is represented with the tag called <ol> and each list item of ordered list is represented with the tag called <li>.

By default the ordered list will displays each list item in 1,2,3,4 format. We can change this by using an attribute called ‘type’.

There are five types of ordered lists available in HTML such as 1 , i , I , a and A.

Example:

```
<ol>
     <li>HTML</li>
     <li>CSS</li>
     <li>JavaScript</li>
     <li>JQuery</li>
     <li>AJAX</li>
     <li>Angular JS</li>
</ol>
```

1. HTML
2. CSS
3. JavaScript
4. JQuery
5. AJAX
6. Bootstrap
7. Angular JS

Example:

```
<ol type="a">
     <li>HTML</li>
     <li>CSS</li>
     <li>JavaScript</li>
     <li>JQuery</li>
     <li>AJAX</li>
     <li>Angular JS</li>
</ol>
```

a. HTML
b. CSS
c. JavaScript
d. JQuery
e. AJAX
f. Bootstrap
g. Angular JS

HTML Image Tag

We can display an image on the webpage by using a HTML tag called <img>.
The <img> tag contains the following attributes,
Src -> to specify the address of an image.
Example:

```
<img src="img/forest.jpg">
```

Alt -> The alt attribute is used to provide an altenate text for an image, if an image is not displayed on the webpage for some reason then the 'alt' will be displayed. If the image is displayed then the alt attribute text will not be displayed.

```
<img src="img/facebook.jpg" width="400" height="200" alt="Facebook Image">
```

Width, height -> normally the image will be displayed of its own width and height, we can customize the width and height using width and height attributes.

```
<img src="img/forest.jpg" width="400" height="200">
```

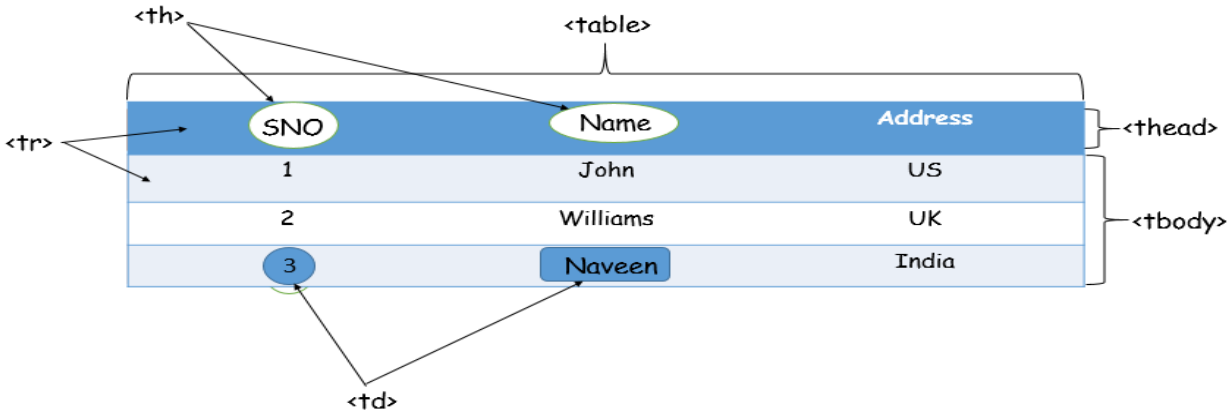
We can even apply a border to an image using CSS as follows,

```
<img src="img/nature_path.jpg" width="400" height="200" id="photo">
```

```
 #photo{
    margin-left: 100px;
    border: 15px solid saddlebrown;
    padding: 5px;
    border-radius: 7px;
}
```

HTML Tables

We can display any content in a table format on the webpage is using a HTML tag called <table>. This is the root tag for the entire table.



In HTML the entire table is represented with the tag called <table>

Each HTML is having two sections such as head section and body section. The Table's head section is represented with the tag called <thead> and the body section of HTML table is represented with the tag called <tbody>.

In HTML table's head section, each cell is represented with the tag called <th> and in the body section each cell is represented with the tag called <td> and finally each row of a HTML table is represented with the tag called <tr>.

Note:

For Each HTML table, defining <thead> and <tbody> is optional. Without these tags also we can be able to display a table. If we are planning to apply any styles related to head and body section of a table, for this we may use <thead> and <tbody> tags.

For any HTML Table by default no borders will be displayed on the browser, we can display the borders using CSS.

Let's understand the HTML code to display the above table on the webpage.

```
<table width="100%">
        <thead>
            <tr>
               <th>SNO</th>
                <th>NAME</th>
                <th>ADDRESS</th>
            </tr>
        </thead>
        <tbody>
            <tr>
                <td>1</td>
                <td>John</td>
                <td>US</td>
            </tr>
            <tr>
                  <td>2</td>
                  <td>Yong</td>
                  <td>Japan</td>
             </tr>
             <tr>
                  <td>3</td>
                  <td>Naveen</td>
                  <td>India</td>
             </tr>
         </tbody>
    </table>
```

Simple Table

SNO	NAME	ADDRESS
1	John	Bangalore
2	Rajan	Hyderabad
3	Williams	Delhi

For every HTML table, we use two special attributes to combine two or more rows or columns.

Colspan

The HTML Colspan attribute used to combine two or more number of columns of a table.
We normally use this for <th> elements.
Let's understand the Colspan,

```html
<table width="100%">
    <thead>
    <tr>
        <th>SNO</th>
        <th colspan="2">NAME & ADDRESS</th>
    </tr>
    </thead>
    <tbody>
    <tr>
        <td>1</td>
        <td>John</td>
        <td>US</td>
    </tr>
    <tr>
        <td>2</td>
        <td>Yong</td>
        <td>Japan</td>
    </tr>
    <tr>
        <td>3</td>
        <td>Naveen</td>
        <td>India</td>
    </tr>
    </tbody>
</table>
```

Table with Col Span

SNO	NAME - ADDRESS	
1	John	Bangalore
2	Rajan	Hyderabad
3	Williams	Delhi

Rowspan

The Rowspan is used to combine two or more number of rows. We normally use this attribute with <td> tag.

Example:

```html
<table width="100%">
    <thead>
    <tr>
        <th>SNO</th>
        <th>NAME</th>
        <th>ADDRESS</th>
    </tr>
    </thead>
    <tbody>
    <tr>
        <td rowspan="2">1</td>
        <td>John</td>
        <td>US</td>
    </tr>
    <tr>
        <td>Yong</td>
        <td>Japan</td>
    </tr>
    <tr>
        <td>2</td>
        <td>Naveen</td>
        <td>India</td>
    </tr>
    </tbody>
</table>
```

Table With Row Span

SNO	NAME	ADDRESS
1	John	Bangalore
	Rajan	Hyderabad
2	Williams	Delhi

Custom Table

We can add few styles for a table to display custom table. Here for this custom table we will apply few styles to the table header such as red color background and white color text as follows,

```
table , th , td{
    border: 2px solid red;
    border-collapse: collapse;
    text-align: center;
    padding: 10px;
}
#custom-table thead{
    background-color: red;
    color: white;
}

<table width="100%" id="custom-table">
   <thead>
       <tr>
           <th>SNO</th>
           <th>NAME</th>
           <th>ADDRESS</th>
       </tr>
   </thead>
   <tbody>
   ...
   </tbody>
<table>
```

Custom Table with CSS

SNO	NAME	ADDRESS
1	John	Bangalore
2	Rajan	Hyderabad
3	Williams	Delhi

HTML Links

In HTML we can convert normal text into a clickable link or hyperlink or hypertext is by using <a> tag.

The hyperlinks are the clickable links and once we click on those links the control will jumps to the specified **href** location.

We identify the hyperlinks with respect to normal text, as for hyperlinks when we hover a mouse on them it automatically turns into a hand symbol.

Not only the text will be converted to hyperlinks, we can even convert an image also a hyperlink and once we click on the image the control will jump to the specified location in the **href** attribute.

Example:

```
<a href="https://www.google.co.in/">Google</a>
```

In the above example the word “Google” is converted as a hyperlink and once we click on this link then control jumps to google website.

We can even give the websites local links for the href attribute of <a> tag so that the control will jump to the local links of the webpage.

Example:

```
<a href="index.html">Home</a>
<a href="contact.html">Contact</a>
```

For HTML links there is an important attribute called 'target' and which specifies where to open the linked document.

The target attribute contains the following values,

_blank - Opens the linked document in a new window or tab

_self - Opens the linked document in the same window/tab as it was clicked (this is default)

_parent - Opens the linked document in the parent frame

_top - Opens the linked document in the full body of the window

```
<a href="https://www.amazon.in/" target="_blank">Amazon</a>
```

We can even convert a normal image as hyperlink as follows,

```
<a href="https://www.youtube.com/" target="_blank">
    <img src="img/youtube.jpg" width="200" height="100">
</a>
```

We even use hyperlinks as a HTML bookmarks.

HTML bookmarks are used to allow readers to jump to specific parts of a Web page. Bookmarks can be useful if your webpage is very long .To make a bookmark, you must first create the bookmark, and then add a link to it. When the link is clicked, the page will scroll to the location with the bookmark.

Example:

First we have to create bookmark with Id attribute and then add an attribute to link to that bookmark to jump to specified part of the webpage.

Example:

```
<h1 id="home">Home Page</h1>
.....
<a href="#home">Go to Home</a>
```

HTML Entities

The HTML Entities are used to display some of the reserved characters and few symbols on the webpage.

For every HTML document it contains some of reserved characters like **<** and **>**.

```
Syntax: &entity_name; or &#entity_name;
```

Symbol	Entity name
<	<
>	>
Copyright symbol	©
Reg Trademark	®
$	$
Rupee Symbol	₹

Example:

```
<h1>This is a &lt; Paragraph &gt;</h1>
```

This is a < Paragraph >

```
<h1>Infosys<sup>&reg;</sup></h1>
```

Infosys®

```
<h1>Copyright &copy; 2018</h1>
```

Copyright © 2018

HTML Iframes:

HTML Iframes are used to display a webpage within another webpage. Using an Iframe a webpage is loaded within then frame of another webpage.

Syntax:

```
<iframe src="URL"></iframe>
```

Each I Frame is having its own width and height, we can specify our own height and width for an iframe.

Ex:

```
<iframe src="08_HTML_Images.html" width="400" height="200"></iframe>
```

Note: We normally use Iframe tag to display a YouTube video on our own webpage.

How to display a YouTube video on our own web page

- ➔ Go to YouTube
- ➔ Select a video to display on the webpage.
- ➔ Click on 'share' link
- ➔ Click on 'Embed' link
- ➔ Select the Embed Options
- ➔ Copy the <iframe> tag completely and paste it in the HTML Document.
- ➔ Open that HTML Document on the browser, the selected YouTube video will be played on our own webpage.

HTML First Mini Project

By using the above learned HTML Tags we can build a simple website. We have built a simple website called "The Great India" website. Please try to develop your own website as per the same architecture given below for "The Great India" Website.

Home | Before & After | Freedom Fighters | Political Leaders | Business Leaders | Sports Leaders

The Great India

Our India

Lorem ipsum dolor sit amet, consectetur adipisicing elit. Aliquid, aut cum facere molestiae nemo nulla pariatur qui rem reprehenderit sint suscipit unde vitae voluptatum! Ad facilis magni molestiae sint voluptates! Aliquid consequuntur doloremque ipsa laboriosam natus neque nesciunt quaerat vel!

India Flag

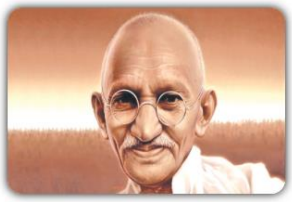
India Map

India Before & After

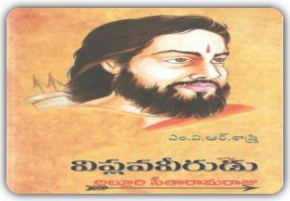
Lorem ipsum dolor sit amet, consectetur adipisicing elit. Aliquam architecto at consectetur cum delectus deserunt distinctio exercitationem expedita fuga libero molestiae natus neque obcaecati, officia officiis provident quae quam quasi quibusdam quis repellat repudiandae sapiente tempora ut vel veniam voluptatem.

Indian Freedom Fighters

Lorem ipsum dolor sit amet, consectetur adipisicing elit. Amet commodi culpa dicta doloremque ducimus eaque inventore, laboriosam maiores nam necessitatibus nisi optio pariatur quasi, quo quod reiciendis repellat similique unde? Blanditiis, dolores ea inventore natus omnis quas repellendus sit ut?



Mahatma Gandhi



Alluri Sita Ramaraju

Subhash Chandra Bose

Bhagat Singh

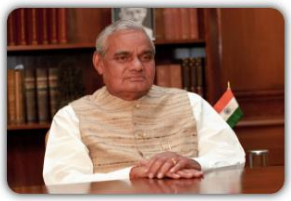
Indian Political Leaders

Lorem ipsum dolor sit amet, consectetur adipisicing elit. Aliquid autem consectetur dolorum eius eos exercitationem fugiat ipsum iste iusto, libero nam, neque nesciunt nisi nobis nulla odio officia placeat possimus quas quasi ratione recusandae saepe tenetur vel voluptates? Culpa, quasi!

Narendra Modi

Lal Bahadur Shastri

Abdul Kalam



AB Vajpai

HTML Forms

A HTML Form is simply an area which contains various form fields. HTML Forms are used to collect the user information and send to a program which is running at server side for further processing.

HTML Form contains various form fields such as Text Box, Password Box, Radio button, Select Box, text area and other input fields. When a user enters / fills in the form data and click on the submit button, then the whole form data will be send to respective server for further processing.

In HTML, the Forms are represented with tag called <form>. This acts as the root tag for all the html form elements.

HTML Forms are divided into 3 categories such as,

1) HTML Form Elements
2) HTML Form Input Types
3) HTML Form Attributes

HTML Form Elements

HTML form elements are also called as HTML Form Tags. These HTML form elements are used to display various form fields on the webpage such as text box, password box, radio button, check box and text area.

Let's discuss some of the HTML form elements as below,

<label> Tag

The HTML <label> tag is used to define a label information for any form field.

Ex:

```
<label for="username">UserName</label>
```

<input> Tag

The <input> tag is used to display almost all the input fields on the webpage such as text box , password box , date selection and range selection.

For Text box,

```
<input type="text" id="username">
```

For Password Box,

```
<input type="password" id="password">
```

Let's design a simple login form using the above tags,

```
<form>
    <label for="username">UserName</label>
    <input type="text" id="username">
    <br>
    <label for="password">Password</label>
    <input type="password" id="password">
    <br>
    <input type="submit">
    <input type="reset">
</form>
```

Note: The 'for' attribute of <label> tag should always be equal to the 'id' attribute of <input> tag.

We can link the <label> and <input> by these attributes only.

If we observe very closely, the above form doesn't have aligned properly. In order to align the form in a proper way we may have to use lot of CSS code. But as now we have not learned the CSS. In order to align the form in proper way is by using a <table> tag.
Let's imagine the above form in the table format. Each line is of a table row. And each form field is in a table data.
Now let's make a new login form with <table> tag for proper alignments.

```
<form>
    <table>
        <tr>
            <td>
                <label>UserName</label>
            </td>
            <td>
                <input type="text">
            </td>
        </tr>
        <tr>
            <td>
                <label>Password</label>
            </td>
            <td>
                <input type="password">
            </td>
        </tr>
        <tr>
            <td>
                <input type="submit" value="Submit">
            </td>
            <td>
                <input type="reset" value="Reset">
            </td>
        </tr>
    </table>
</form>
```

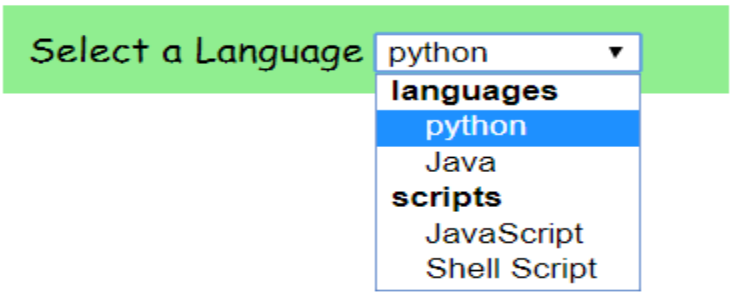
<Select> , <option> Tags

The <select> Tag is used to display a select box on the webpage. This is also called as a dropdown list. And each option inside of a select box is represented by <option> tag.
Let's understand the select box with an example,

```
<form>
    <label>Select a Language</label>
    <select>
        <option>HTML</option>
        <option>CSS</option>
        <option>JavaScript</option>
        <option>Bootstrap</option>
        <option>Angular JS</option>
    </select>
</form>
```

We can even group the options inside the select box using <optgroup> tag.
Let's discuss the example of <optgroup> tag

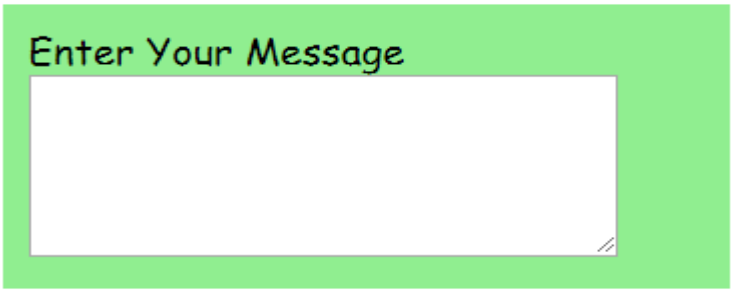
```
<form>
    <label>Select a Language</label>
    <select>
        <optgroup label="languages">
            <option>python</option>
            <option>Java</option>
        </optgroup>
        <optgroup label="scripts">
            <option>JavaScript</option>
            <option>Shell Script</option>
        </optgroup>
    </select>
</form>
```



<textarea> tag

The HTML <textarea> tag is used to display a message box or comments box on the webpage.
This <textarea> tag contains two mandatory attributes such as rows, cols.
'rows' indicates the height of textarea and 'cols' indicates the width of textarea box.
Let's discuss the example of Comments box using <textarea> tag as follows,

```
<form>
    <label for="comments">Enter Your Message</label>
    <textarea rows="5" cols="30" id="comments"></textarea>
</form>
```



<datalist> tag

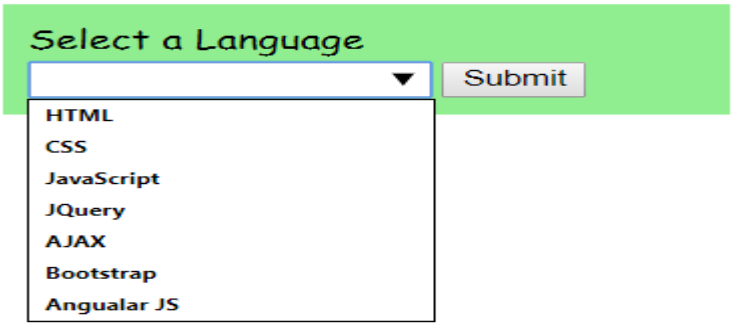
This tag is introduced in the HTML5 version.

The <datalist> element specifies a list of pre-defined options for an <input> element.

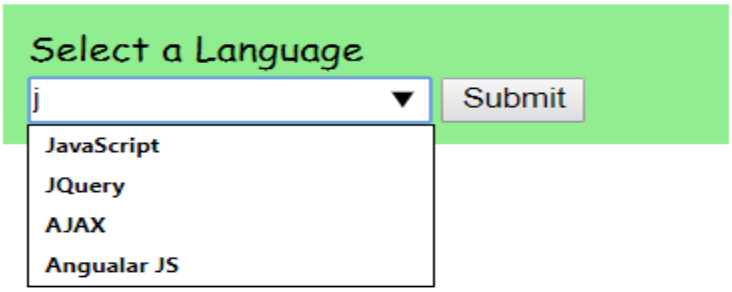
Users will see a drop-down list of the pre-defined options as they input data.

The list attribute of the <input> element, must refer to the id attribute of the <datalist> element.

```
<form>
    <label for="technologies">Select a Language</label>
    <input list="technologies">
    <datalist id="technologies">
        <option>HTML</option>
        <option>CSS</option>
        <option>JavaScript</option>
        <option>JQuery</option>
        <option>AJAX</option>
        <option>Bootstrap</option>
        <option>Angualar JS</option>
    </datalist>
    <input type="submit" value="Submit">
</form>
```



Note: This <datalist> is the more refined version of select box. Here we will get the options to be displayed based on the search query in datalist box.



HTML Form Input Types

The HTML Form input types are used to display various input fields with only one tag called <input>

In order to display various input fields we use only an <input> tag but the extra 'type' attribute value changes. Depend on the 'type' attribute value we will different input field.

type="text"

This is used to display a text box on the webpage.

Example

```
<form>
    <label>User Name</label>
    <input type="text">
</form>
```

type="password"

This attribute is used to display a password box on the webpage.

Example:

```
<form>
    <label>Password</label>
    <input type="password">
</form>
```

type="submit"

This Attribute is used to display a submit button on the webpage.

Once a user enters the complete form data and clicks on the submit button then the entire form data will be sent the program which is running on the server.

type="reset"

This attribute is used to display a reset button on the webpage.

This button will reset the entire form. If any input field is filled by the user will get cleared with reset button.

type="radio"

This attribute is used to display a radio button on the webpage. The radio button is used to select only one input choice among many number of choices displayed.

Example:

```
<form>
    <label>Is HTML Easy ?</label>
    <input type="radio" name="easy" value="yes">YES
    <input type="radio" name="easy" value="no">NO
</form>
```

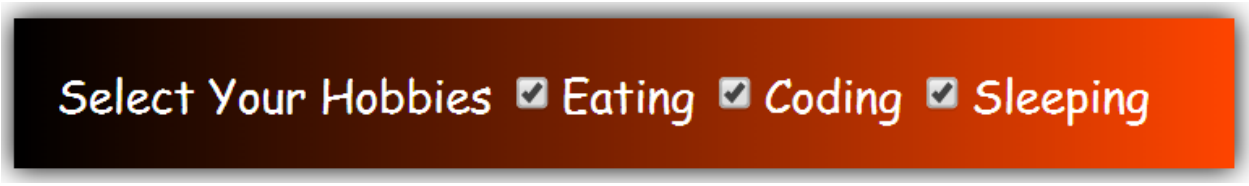
Note: Here the 'name' attribute is mandatory, and the value of name attribute should be same for all the radio buttons.

type="checkbox"

This attribute is used to display a checkbox on the webpage. This checkbox lets the user to select zero or more number of choices among the displayed check boxes.

Example:

```
<form>
    <label>Select Your Hobbies</label>
    <input type="checkbox" value="eating">Eating
    <input type="checkbox" value="coding">Coding
    <input type="checkbox" value="sleeping">Sleeping
</form>
```



HTML 5 Input Types

The following are from HTML 5 version of Input Types,

type="color"

This is used for color selection among various colors on the webpage.

Example:

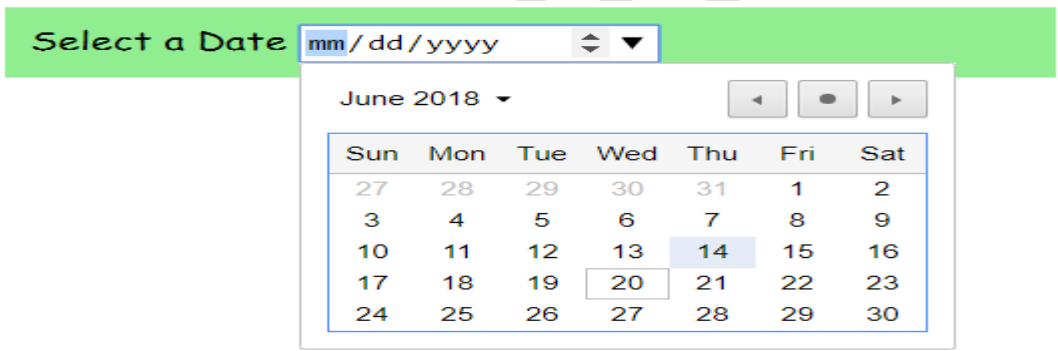
```
<form>
    <label>Select a Color</label>
    <input type="color">
</form>
```

type="date"

This attribute is used for date selection. Here it will display a calendar to select a date.

Example:

```
<form>
    <label>Select a Date</label>
    <input type="date">
</form>
```

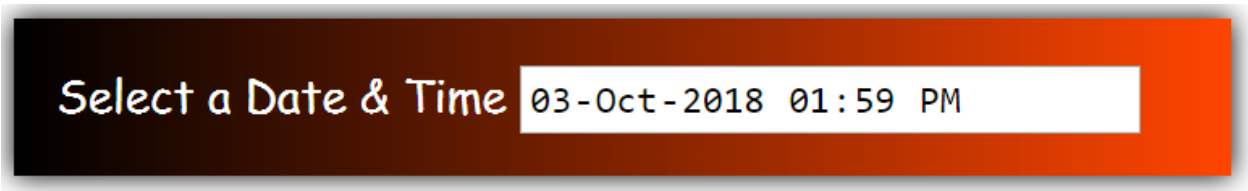


type="datetime-local"

This attribute is used to select a date as well as time. This is date and time selection box.

Example:

```
<form>
    <label for="datetime">Select a Date & Time</label>
    <input type="datetime-local" id="datetime">
</form>
```



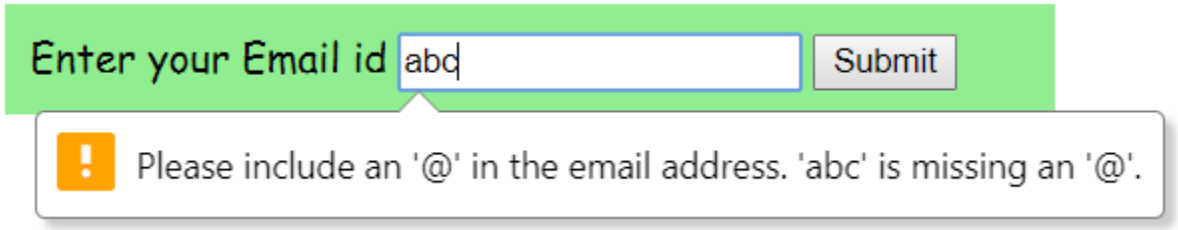
type="email"

This is used to display an email box. This seems like a normal text box but in the backend there is a email validation will be happened by HTML5.

It uses a pattern for default email validation using HTML5.

Example:

```
<form>
    <label>Enter your Email id</label>
    <input type="email">
    <input type="submit" value="Submit">
</form>
```



type="month"

This attribute is used for month selection. This will also display a calendar but here we can be able to select a specific month from that calendar.

Example:

```
<form>
    <label for="month">Select a Month</label>
    <input type="month" id="month">
</form>
```

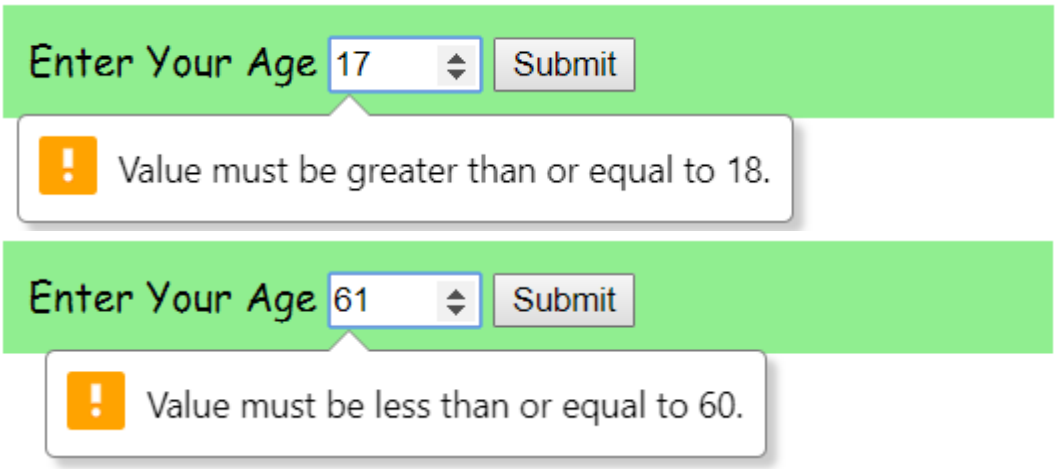
type="number"

This is used for number selection box. Here we can enter only numbers and we cannot be able to enter any other character other than number except ‘e’ (for exponential values).

Here also some default validation will be happened for number, we can specify minimum and maximum numbers validation.

Example:

```
<form>
    <label>Enter Your Age</label>
    <input type="number" min="18" max="60">
    <input type="submit" value="Submit">
</form>
```

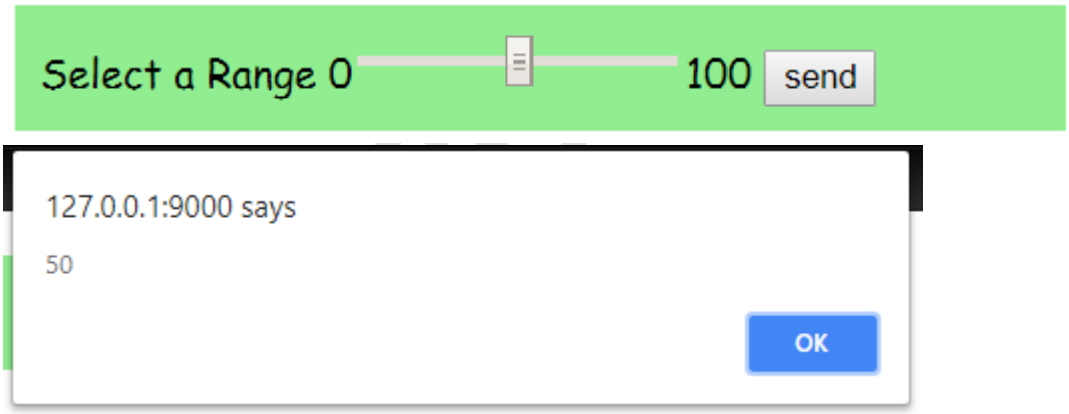


type="range"

This is used for range selection. Here also we can be able to provide min and max value for the range.

Example:

```
<form>
    <label>Select a Range</label>
    0<input type="range" min="0" max="100" id="some">100
    <input type="button" value="send"
           onclick="alert(document.getElementById('some').value)">
</form>
```

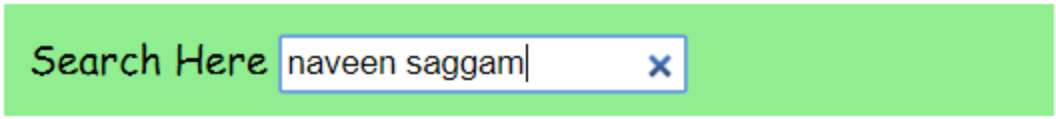


type="search"

This input type is used to display a search box on the webpage. This is almost same as text box but a little difference is here in the search box we will get a cross symbol to reset the search query

Example:

```
<form>
    <label for="search">Search Here</label>
    <input type="search" id="search">
</form>
```



type="time"

This input type is used for time selection on the webpage. Here no clock will be displayed by default. We needs to enter the time manually.

Example:

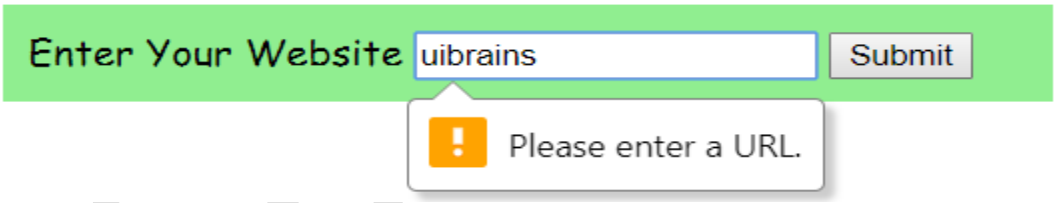
```
<form>
    <label for="time">Select a Time</label>
    <input type="time" id="time">
</form>
```

type="url"

This input type is used for url validation. Here they use a default validation for url selection. It expects pattern of 'http://'.

Example:

```
<form>
    <label for="url">Enter Your Website</label>
    <input type="url" id="url">
    <input type="submit" value="Submit">
</form>
```



HTML Input Attributes

In HTML Forms there are some special attributes are available to apply the special effects for the input fields on the webpage like to disable the form fields , or to make a form field read-only or to make a form field mandatory field.

The HTML Form Input Attributes are as follows,

'value' Attribute

Each HTML Form field is having a 'value' attribute, this is having connection with the server side program.

Once the user fills in the form, then this value attribute holds the actual value of that specific form field.

```
<form>
    <label>User Name</label>
    <input type="text" value="Naveen">
</form>
```

Input Attribute 'Value'

'readonly' Attribute

This attribute is used to make any input field as read-only field. If any input field is declared as with this attribute, then that input field is unmodifiable.

```
<!-- Input Attribute readonly --->
<h2>Input Attribute 'readonly'</h2>
<form>
    <label>User Name</label>
    <input type="text" value="Naveen" readonly>
</form>
```

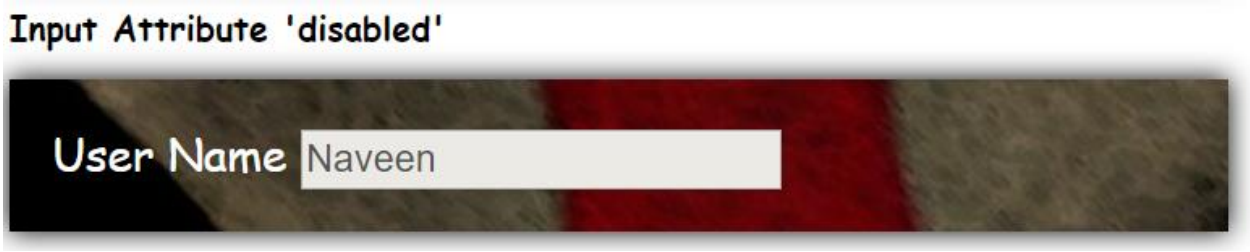
Input Attribute 'readonly'

'disabled' Attribute

This attribute is used to disable an input field.

This is same read-only, but which is better option than read-only

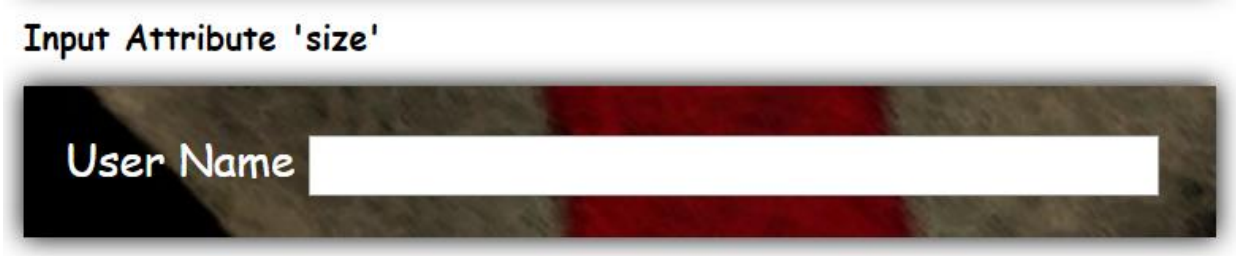
```
<!-- Input Attribute disabled --->
<h2>Input Attribute 'disabled'</h2>
<form>
    <label>User Name</label>
    <input type="text" value="Naveen" disabled>
</form>
```



'size' Attribute

This attribute is used to increase or decrease the size of any specific input field.
We normally use this to increase or decrease the size of a textbox or a password box.

```
<!-- Input Attribute size --->
<h2>Input Attribute 'size'</h2>
<form>
    <label>User Name</label>
    <input type="text" size="40">
</form>
```



'rows' , 'cols' Attributes

These attributes are used with the <textarea> tag.
Here 'rows' indicates height of textarea box
'cols' indicates width of textarea box.

```
 <!-- Input Attribute rows , cols --->
<h2>Input Attribute 'rows' , 'cols'</h2>
<form>
    <label>Your Message</label>
    <textarea rows="5" cols="30"></textarea>
</form>
```

HTML5 Form Input Attributes

In HTML5 version 'w3c.org' introduced the new attributes for input fields, they are as follows,

'required' attribute

By default, all the input fields are optional fields in the HTML Form.
We can convert them as mandatory fields or required fields using this attribute.

```
<h2>Input Attribute 'required'</h2>
<form>
    <label>User Name *</label>
    <input type="text" required>
    <input type="submit" value="Submit">
</form>
```

'novalidate' Attribute

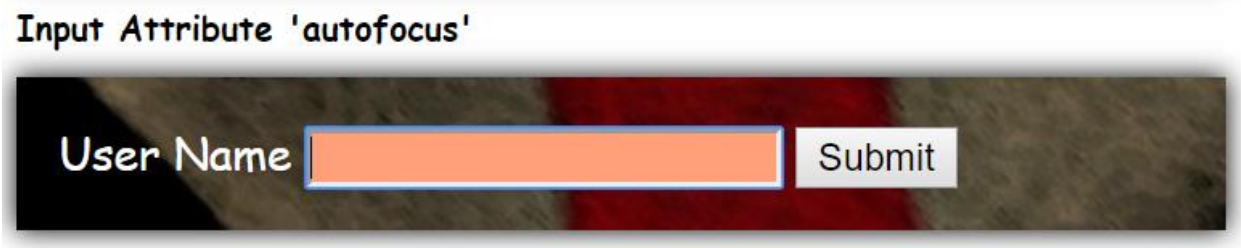
By default most of the HTML5 form fields are having its own validation messages, in order to disable the default validation messages we have to use this attribute.
We normally use this, while applying our own validation messages using JavaScript, JQuery, and Angular JS.
We have to use this attribute for the <form> itself.

```
<h2>Input Attribute 'novalidate'</h2>
<form novalidate>
    <label>Enter Your Website</label>
    <input type="url">
    <input type="submit" value="Submit">
</form>
```

'autofocus' Attribute

By default, none of the form field is automatically focused.
In order to make any form field on the webpage to automatically focused we have to use this attribute.

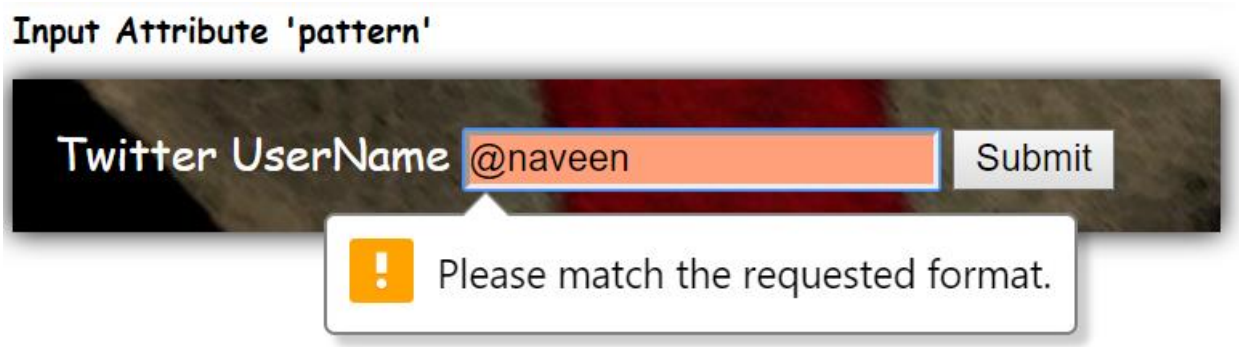
```
<h2>Input Attribute 'autofocus'</h2>
<form>
    <label>User Name</label>
    <input type="text" autofocus>
    <input type="submit" value="Submit">
</form>
```



'pattern' attribute

Most of the HTML5 Form fields are using its own default pattern.
We override the default pattern of each form fields using this attribute.

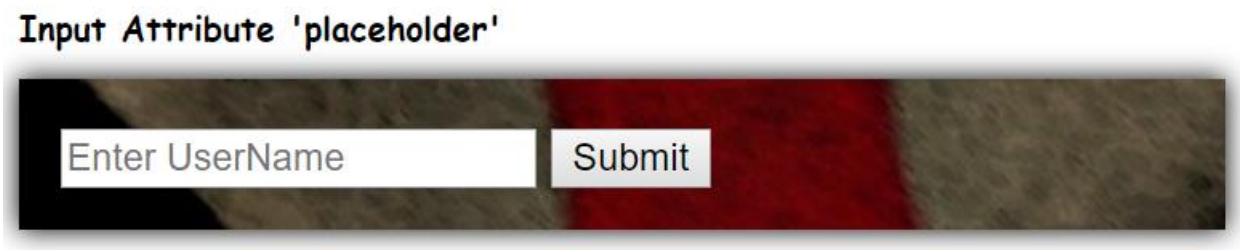
```
<h2>Input Attribute 'pattern'</h2>
<form>
    <label>Twitter UserName</label>
    <input type="text" pattern="^[A-Za-z0-9_$]{1,15}$">
    <input type="submit" value="Submit">
</form>
```



'placeholder' Attribute

This is a replacement of the <label> tag.
Here the label text will be displayed inside the input field itself.

```
<h2>Input Attribute 'placeholder'</h2>
<form>
    <input type="text" placeholder="Enter UserName">
    <input type="submit" value="Submit">
</form>
```



Online Job Application Form

We can develop a simple webpage using all the above HTML form tags, which is an Online Job Application form as follows,

Online Job Application Form

Personal Information

FirstName | Middle Name | Last Name

Father's Name | Mother's Name

Current Address | Permanent Address

Phone Numbers

Landline | Mobile Number

Date Of Birth dd-mm-yyyy | Place Of Birth

Select Gender ● Male ● Female

Highest Qualification | Year Of Passing --------, ----

Languages Known ■ Telugu ■ Hindi ■ English

Select Your Hobbies ■ Eating ■ Coding ■ Sleeping

About Yourself

Nationality American

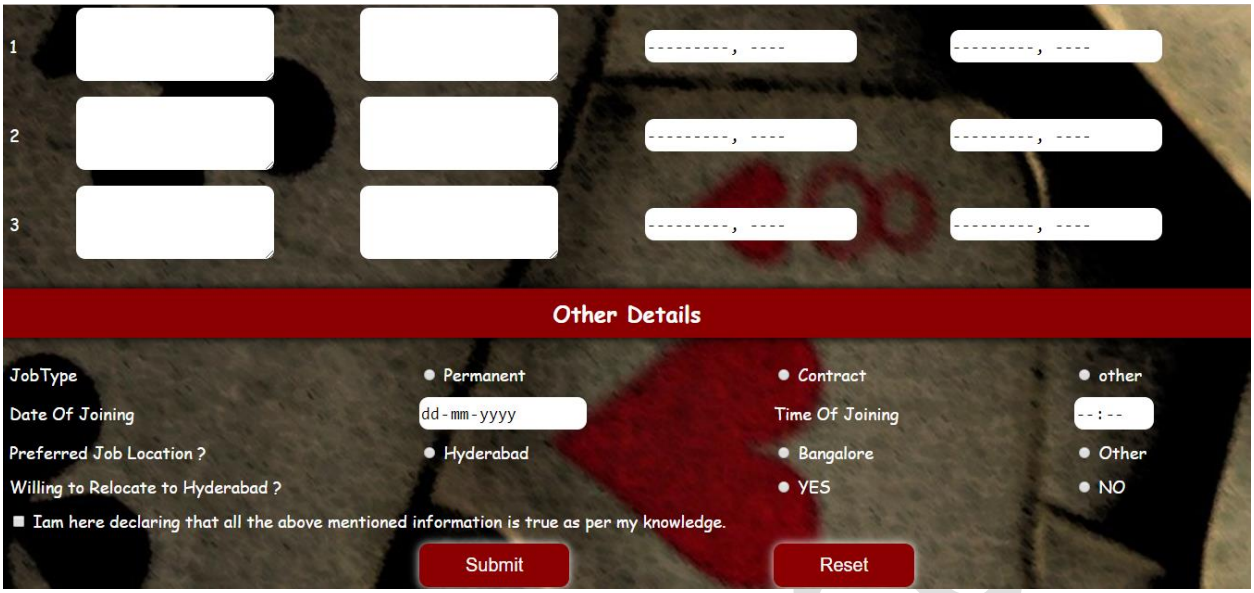
AadharCard Number | PANCARD Number

Education Details

SNO	Qualification	Institute / University	Year Of Passing	Marks(%)
1			--------, ----	
2			--------, ----	
3			--------, ----	

Work Experience

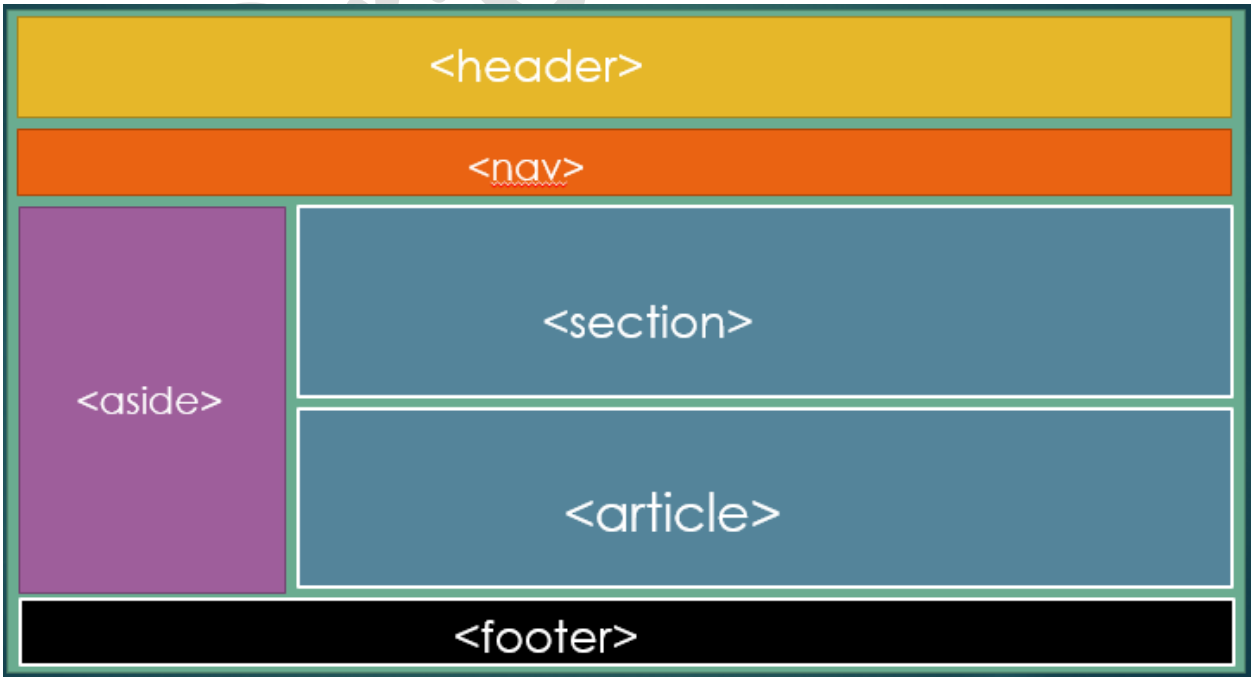
SNO	Company Address	Work / Roles	From	To



HTML5 Layout elements

In HTML5 Version, 'w3c.org' introduced new tags for layout of any webpage.
These HTML tags are used to structure the webpage in a proper manner. These are just structural elements, and just by applying them no styles will be applied by default.

We have to apply the styles with the concept called CSS.



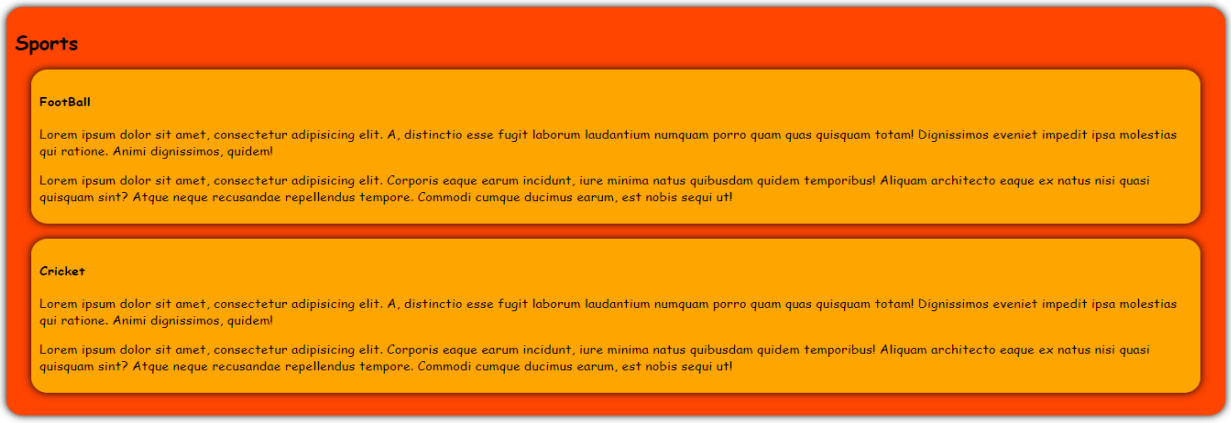
HTML <section> Tag

The <section> tag is used to divide the webpage into various sections.
A home page could normally be split into various sections for introduction, content, and contact information.

HTML <article> Tag

The <article> Tag specifies independent, self-contained content.
This is used to write any kind of articles in the webpage like news article, Blog posts and etc.
This can be inside <section> Tag
Examples :
Forum post
Blog post
Newspaper article

```
<!-- Sports Section -->
<section>
    <h2>Sports</h2>
    <article>
        <h4>FootBall</h4>
        <p>Lorem ipsum dolor sit amet... </p>
        <p>Lorem ipsum dolor sit amet... </p>
    </article>
    <article>
        <h4>Cricket</h4>
        <p>Lorem ipsum dolor sit amet...</p>
        <p>Lorem ipsum dolor sit amet...</p>
    </article>
</section>
```



HTML <header> Tag

The <header> tag specifies the main header for the webpage.
In this <header> tag, we can display the company Logo and company information and any subtitle of the company.
We normally display the Header information at the top section of the webpage

HTML <footer>Tag

The <footer> element specifies a footer for a webpage.
A footer typically contains the author of the document, copyright information, links to terms of use, contact information, etc.

```
<header>
    <h2>UiBrains Technologies</h2>
    <p>The Smart Way of Learning</p>
</header>

<section>
    <h2>Offline Training</h2>
    <article>
        <h4>UI Technologies</h4>
        <p>Lorem ipsum dolor sit...</p>
        <p>Lorem ipsum dolor sit...</p>
    </article>
</section>

<footer>
    <h3>Copyright &copy; 2018 , UiBrains Technologies</h3>
    <p>All Rights Reserved</p>
    <p>Developed & Maintained by Naveen Saggam</p>
</footer>
```

Online Training

UI Technologies

Lorem ipsum dolor sit amet, consectetur adipisicing elit. Ad commodi consequuntur debitis, earum, eius est eum ex expedita explicabo hic inventore labore molestias nihil nobis pariatur quod rerum suscipit tenetur totam vel? Doloremque in minima nisi odio, officia quo veritatis.

Lorem ipsum dolor sit amet, consectetur adipisicing elit. Autem consequuntur cumque, cupiditate ea ex magni maxime minima minus modi nihil, nisi perspiciatis tempora voluptatum. Blanditiis deleniti facere natus? Aliquam animi blanditiis consectetur dicta dolor dolore dolores eius explicabo fugit harum id impedit nulla, pariatur perferendis quod ratione sapiente, velit voluptatibus.

MEAN Stack

Lorem ipsum dolor sit amet, consectetur adipisicing elit. Ad commodi consequuntur debitis, earum, eius est eum ex expedita explicabo hic inventore labore molestias nihil nobis pariatur quod rerum suscipit tenetur totam vel? Doloremque in minima nisi odio, officia quo veritatis.

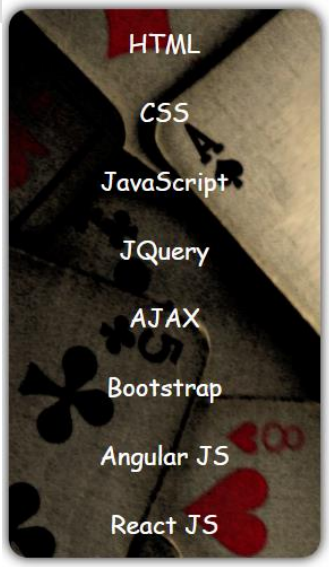
Lorem ipsum dolor sit amet, consectetur adipisicing elit. Autem consequuntur cumque, cupiditate ea ex magni maxime minima minus modi nihil, nisi perspiciatis tempora voluptatum. Blanditiis deleniti facere natus? Aliquam animi blanditiis consectetur dicta dolor dolore dolores eius explicabo fugit harum id impedit nulla, pariatur perferendis quod ratione sapiente, velit voluptatibus.

Copyright © 2018 , UiBrains Technologies
All Rights Reserved
Developed & Maintained by Naveen Saggam

HTML <aside> Tag

The <aside> tag defines some content which is aside from the content it is placed in (like a sidebar).
The aside content should be related to the surrounding content.

```
<aside>
    <ul>
        <li>
            <a href="#">HTML</a>
        </li>
        <li>
            <a href="#">CSS</a>
        </li>
        <li>
            <a href="#">JavaScript</a>
        </li>
          ..........
    </ul>
</aside>
```



HTML <nav> Tag

The <nav> tag defines a set of navigation links.
We normally use this <nav> tag to display a websites main navigation links on the top of the webpage.
<nav> tag is used mostly for top navigation purpose only.

```
<nav>
    <ul>
        <li>
            <a href="#" class="active">Home</a>
        </li>
        <li>
            <a href="#" >About Us</a>
        </li>
        <li>
            <a href="#">Contact Us</a>
        </li>

         <li>
             <a href="#">Careers</a>
         </li>
         <li>
             <a href="#">Forum</a>
         </li>
    </ul>
</nav>
```

Home | About Us | Contact Us | Courses | Careers | Forum

<figure>, <figcaption> Tags

The <figure> tag is used to display an image in the gallery format.
The purpose of a <figcaption> element is to add a visual explanation to an image.
In HTML5, an <img> and a <figcaption> can be grouped together in a <figure> element.

```
<figure>
    <img src="img/nature_path.jpg">
    <figcaption>Nature Path</figcaption>
</figure>
```

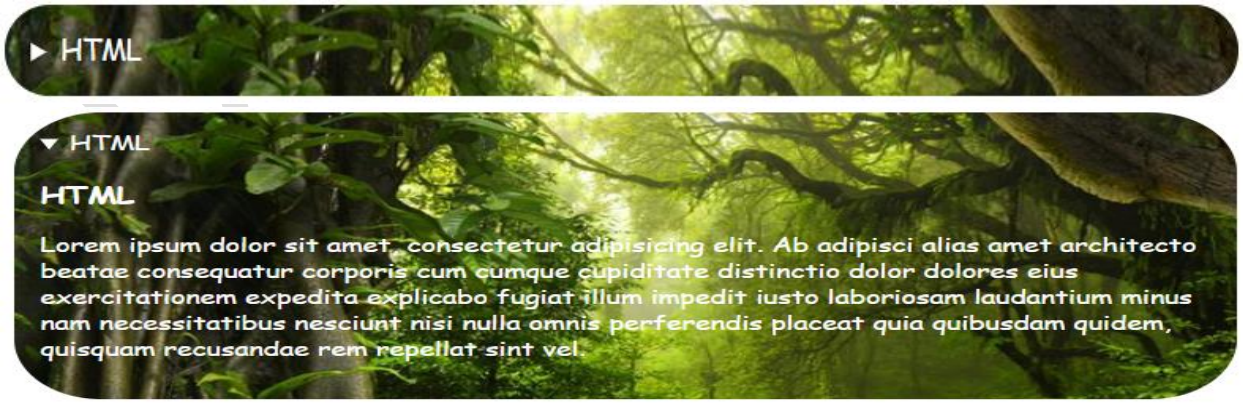
HTML <details> Tag

The <details> tag specifies additional details that the user can view or hide on demand.
The <details> tag can be used to create an interactive widget that the user can open and close.

HTML <summary> Tag

The <summary> tag defines a visible heading for the <details> element.
The heading can be clicked to view/hide the details.

```
<details>
    <summary>HTML</summary>
    <div>
        <h3>HTML</h3>
        <p>Lorem ipsum dolor sit ...</p>
    </div>
</details>
```



HTML <main> tag

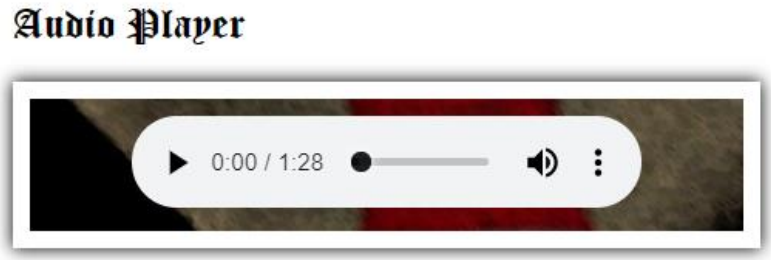
The <main> tag specifies the main content of a webpage.
The content inside the <main> element should be unique to the webpage and not repeated anywhere in the webpage.

```
<main>
    <h1>UiBrains Technologies</h1>
    <p>Lorem ipsum dolor sit amet....</p>
</main>
```

HTML <audio> Tag

HTML <audio> tag is used to play an audio the webpage.
This file loacation will be loaded from the local source.
Most preferred file format for an audio is '.mp3'

```
<div>
   <audio controls>
       <source src="media/nature.mp3" type="audio/mp3">
   </audio>
</div>
```



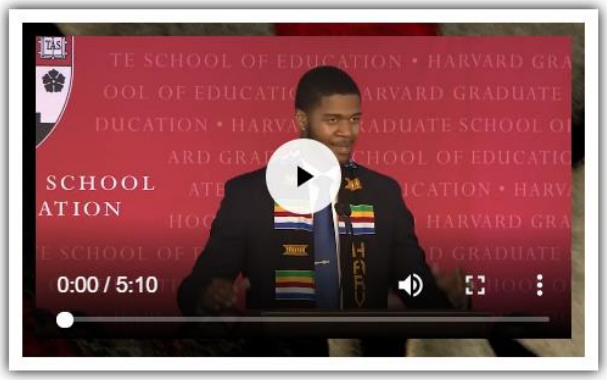
HTML <video> Tag

HTML <video> tag is used to play a video the webpage.
This file loacation will be loaded from the local source.

Most preferred file format for an video is '.mp4'

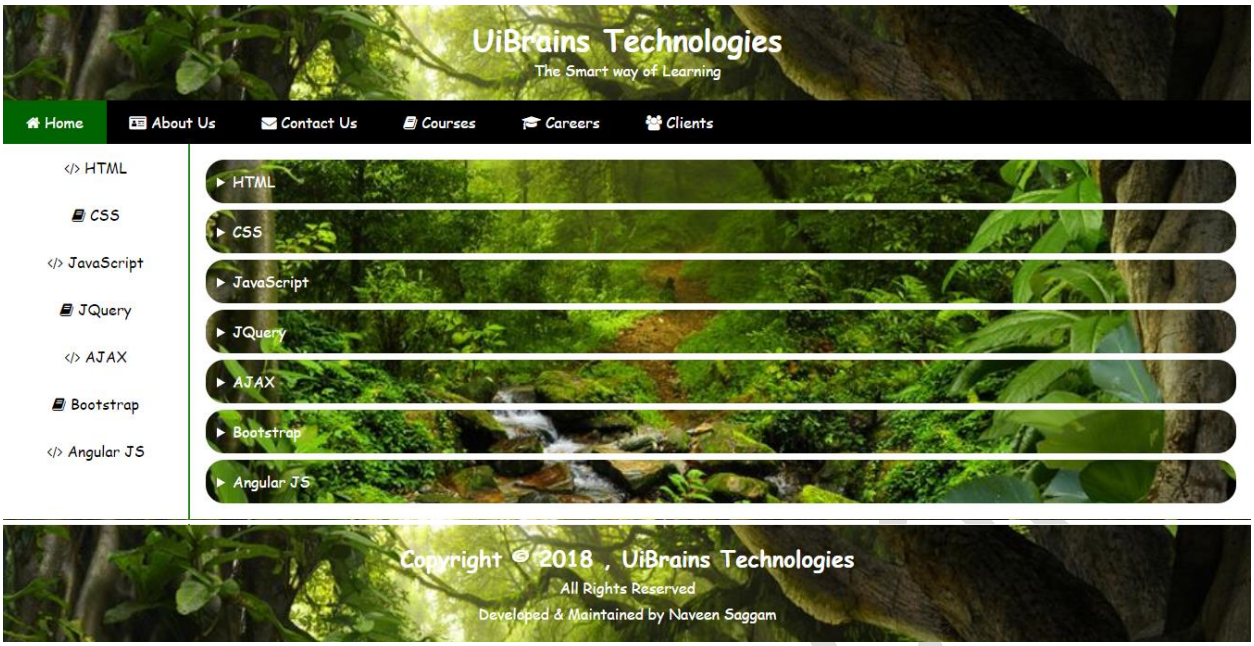
```
<h2>Video Player</h2>
<div>
    <video controls>
        <source src="media/harvard.mp4" type="video/mp4">
    </video>
</div>
```

Video Player



Company Website

We can build a simple website by using all the above HTML 5 Layout tags and which is as follows,



UiBrains.com

Cascading Style Sheet

Introduction to CSS

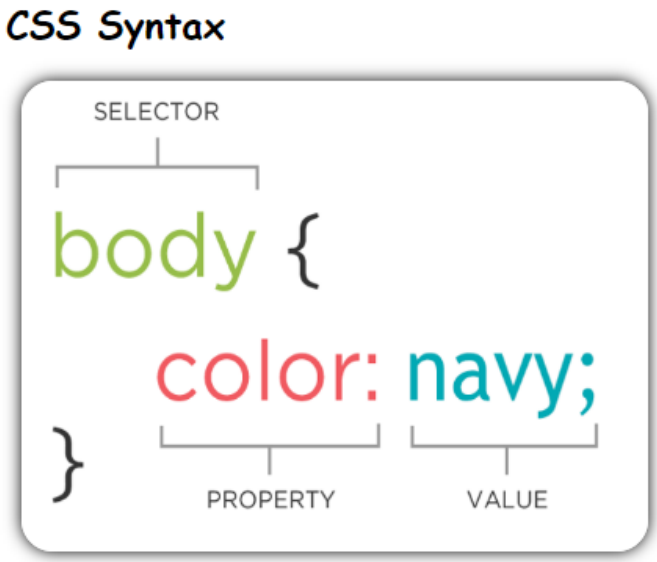
To develop any website we required majorly 3 technologies like HTML, CSS, and JavaScript. The HTML Language is just to provide the structure of the webpage with some base styles to display a content on the webpage.

The CSS Stands for Cascading Style Sheet.

We use CSS to override the default styles of HTML and apply various extra styles like padding, margin, colors, transformation and animations.

CSS describes how to present the data on the webpage with user defined styles.

We can always use CSS along with the HTML language only. First with the HTML we will define the structure and by using CSS we will select each HTML element and apply styles to it.



We can use the CSS with a HTML in three ways, such as

1) Inline Approach
2) Internal Approach
3) External Approach

Inline Approach

In this process of applying styles to HTML Elements we use 'style' Attribute.

To apply the text color red for a paragraph using inline approach is,

```
<p style="color: red">Lorem ipsum dolor sit...</p>
```

In this approach, the styles will be applied for a specific tag only. In order to apply same styles for multiple paragraphs, we may have to duplicate the styles as follows,

```
<p style="color: red">Lorem ipsum dolor sit...</p>
<p style="color: red">Lorem ipsum dolor sit...</p>
<p style="color: red">Lorem ipsum dolor sit...</p>
<p style="color: red">Lorem ipsum dolor sit...</p>
<p style="color: red">Lorem ipsum dolor sit...</p>
```

In this approach there is no single place or selector to change styles of all the paragraphs.

Drawbacks

This approach is only used to apply styles for a specific tag only, if we wants to apply the styles we have to repeat the same styles all the required HTML Tags.

This approach increases the code duplication and code redundancy.
This increase code maintainability of the application.

In ordered to resolve this we have to internal approach.

Internal Approach

In this approach we will apply the CSS styles using HTML <style> tag. We have add this <style> tag in the <head> section of any HTML Document.

To apply the text color red for all the paragraphs of a HTML Document is as follows,

```
<style>
    p{
        color: red;
    }
</style>

<p>Lorem ipsum dolor sit amet.</p>
<p>Lorem ipsum dolor sit amet.</p>
<p>Lorem ipsum dolor sit amet.</p>
<p>Lorem ipsum dolor sit amet.</p>
<p>Lorem ipsum dolor sit amet.</p>
```

In the above code, we selected a ‘p’ tag and applied styles to it. In this approach all the paragraphs in this HTML document got the styles.

Drawbacks

But in this approach we can apply the styles only for the specific HTML Document, these styles will not be applied to any other HTML Document

In ordered to apply the same styles to all the HTML pages in a website, we may to again duplicate the styles with all the HTML documents.

This increases the code duplication and code redundancy.

This approach increase the maintainability of the application.

In order to resolve this we have to use External approach.

External Approach

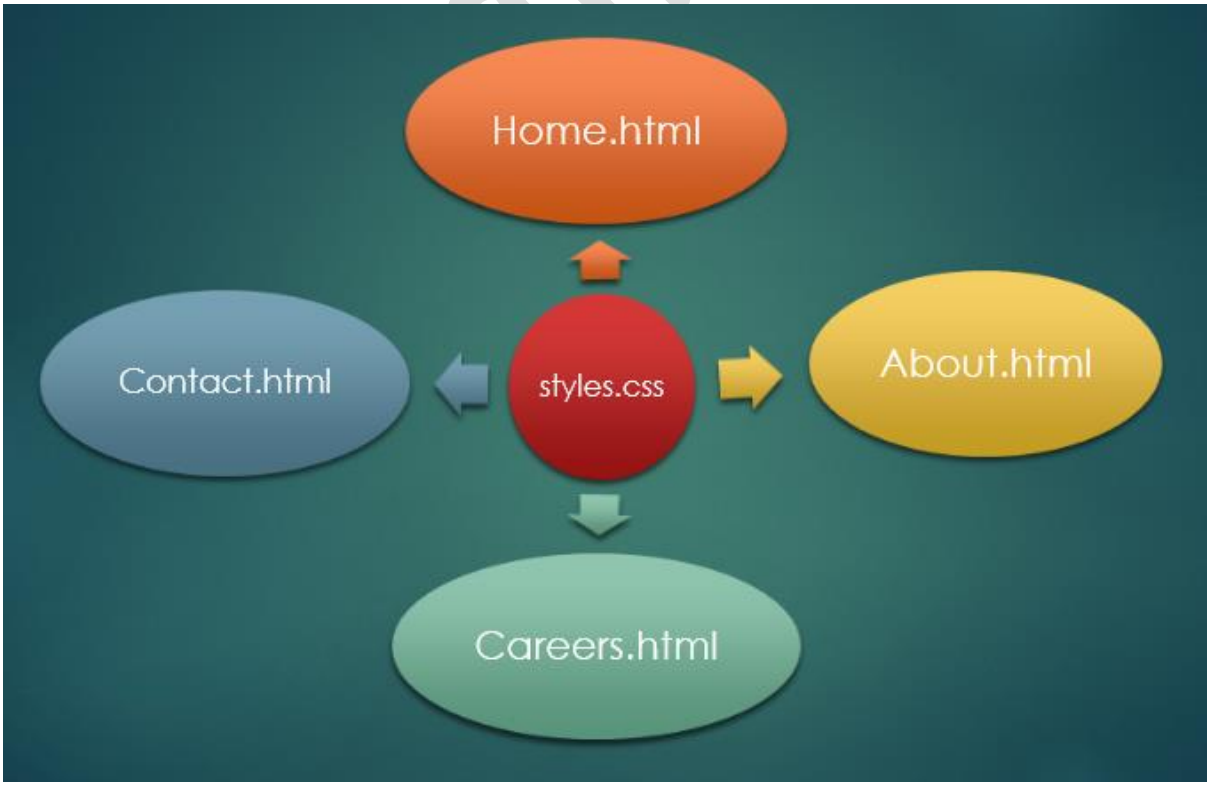
This is the best approach to apply styles to any webpage. In this approach we will create a separate '.css' file and apply all the styles to it.

We can connect this '.css' file to any HTML document using a HTML <link> Tag.

Ex:

```
<link rel="stylesheet" href="styles.css">
```

We have to add this link to the head section of all the HTM documents of a website.



We normally use CSS to select a specific HTML element and apply styles to it. To select any HTML element we use the following CSS Selectors.

CSS Selectors

The CSS Selectors are classified as follows,

1) Element Selector / Tag Selector
2) Id Selector
3) Class Selector
4) Attribute Selector
5) Pseudo selector

Element Selector

In this selector we will select a specific element or tag and apply styles to it.

To apply the background-color red for a specific tag or element is as follows,

```
p{
    background-color: red;
}

<p>Lorem ipsum dolor sit amet.</p>
<p>Lorem ipsum dolor sit amet.</p>
<p>Lorem ipsum dolor sit amet.</p>
<p>Lorem ipsum dolor sit amet.</p>
```

Id Selector

In the above example, we can apply the styles for all the Elements in a HTML document.

In order to apply some styles for a specific element, we have to give a unique id and we select this id and apply styles to it.

```
#green{
    background-color: green;
    color: white;
}
<p>Lorem ipsum dolor sit amet.</p>
<p id="green">Lorem ipsum dolor sit amet.</p>
<p>Lorem ipsum dolor sit amet.</p>
```

Note: The id attribute for any HTML element is unique for the entire HTML document. If any element is defined with an id, then the same id cannot be repeated anywhere else in the same Document.

Class selector

In the above example, we can apply styles for any specific HTML Element with an Id. In order to apply styles to a group of HTML elements we should use class selector.

In this class selector we can add the same class name for a group of elements and apply styles to it.

To apply the background color green for a group of HTML elements using Class selector as follows,

```
.green{
   background-color: green;
   color: white;
}
<p>Lorem ipsum dolor sit amet.</p>
<p class="green">Lorem ipsum dolor sit amet.</p>
<p>Lorem ipsum dolor sit amet.</p>
<h3 class="green">Lorem ipsum dolor sit amet.</h3>
```

Attribute Selector

By using CSS we can even select a specific attribute of a HTML Element and apply styles to it. These styles are used when we apply some styles only for a text box or password box.

As both are using an <input> tag, we can differentiate them using an attribute only.

To apply the background color green for a text box using attribute selector as follows,

```
input[type='text']{
    background-color: forestgreen;
}
```

To apply background color red for a password box using attribute selector as follows,

```
input[type='password']{
    background-color: red;
}
```

Pseudo Selector

An action which a user is performing on the webpage is called as an event.

This is the selector to apply the CSS styles for some special events on the browser.

To apply the background color red which focus on a text box is as follows,

```
input[type='text']:focus{
    background-color: red;
}
```

To apply the background color green while we hover on a button using pseudo selector is as follows

```
input[type='submit']:hover{
    background-color: forestgreen;
}
```

CSS Colors

To specify a color for any HTML element we use various color values. But each color we have many number of shades, in order to specify all shades using normal text color notation is highly impossible. Using CSS Color values we can specify all the shades of a color.

CSS colors contains the following values.

Textual color

In this we will specify the color values using textual format. Using this way we can use a few color values only. We cannot be able to specify the exact shade of the required color.

```
background-color: green;
```

RGB(red,green,blue)

In this approach of providing the color values, we will use various combinations of color values of red, green, blue.

Here the color values starts from 0 – 255.

```
background-color: rgb(0,0,0); // black color
color: rgb(255,255,255); // white color
```

HEX (Hexa decimal)

In this approach we will specify the color values using hexadecimal format of color values with the combination of red, green, blue.

Here the color values starts from 00 – FF.

```
background-color: #000000; // black color
color: #FFFFFF; // white color
```

```
#hex{
    background-color: #4267B2;
}
```

HSL (Hue, saturation, Lightness)

This is another way of providing the color values for any HTML element.

```
syntax: hsl(hue,saturation,lightness);
```

Here,

Hue -> a Degree on the color wheel

(0:red , 120:green: , 240: blue , 360:black)

Saturation -> is a percentage value

(0% means a shade of gray, and 100% is the full color)

Lightness -> is also a percentage

(0% is black, 50% is neither light nor dark, 100% is white)

```
#hsl{
    background-color: hsl(220,45%,47%);
}
```

RGBA (red, green, blue, alpha)

This is exactly same as RGB with an extra parameter called 'alpha'.

```
syntax: rgba(red,green,blue,alpha);
```

Here 'alpha' indicates the transparent value , its value between 0 & 1.

0 -> Full Transparent

1 -> Full Color

```
#rgba{
    background-color: rgba(66,103,178,0.5);
}
```

HSLA (Hue, Saturation, lightness, alpha)

This is exactly same as HSL with an extra parameter 'alpha'

```
Syntax: hsla(Hue ,Saturation,lightness,alpha);
```

Here 'alpha' indicates the transparent value , its value between 0 & 1.

0 -> Full Transparent

1 -> Full Color

```
#hsla{
    background-color: hsla(220,45%,47%,0.5);
}
```

CSS Backgrounds

The CSS backgrounds properties are used to apply various background styles for the HTML Elements.

The CSS Backgrounds contains the following properties,

background-color

This property is used to apply any background-color for the HTML elements using textual color , RGB , HEX , HSL , RGBA , HSLA values.

```
background-color: rgb(0,0,0); // black color
color: rgb(255,255,255); // white color
```

background-image

This property is used to apply the background-image for any HTML element.

```
background-image: url("img/image5.jpeg");
```

background-repeat

This property is used whether to repeat the image in the background or nor-repeat.

this property contains the following values,

repeat-x -> to repeat the image in X-axis

repeat-y -> to repeat the image in Y-axis

no-repeat -> to display a single image

```
background-repeat: no-repeat;
```

background-attachment

This property is used to fix the image in the background or scroll the image along with the scroll bar.

This property contains the following values

fixed - > to fix the image in the background

scroll -> to scroll the image along with the scroll bar.

```
background-attachment: fixed;
```

background-position

this property is used to place the image in the specified position like right , top , left , bottom , center.

```
background-attachment: fixed;
```

Shortcut property for background styles

we can apply all the background properties using a single CSS property called 'background'

```
syntax: background:color url repeat attachment position;
Ex: background-color:white url('img.abc.jpg') no-repeat fixed bottom;
```

CSS Borders

CSS border are used to apply various border styles for any HTML Elements. To apply a simple border for any HTML Element we use CSS borders.

In CSS borders we have the following properties

border-width

This property indicates the width of the border. We can specify the width of a border in px, cm, mm, em, inch etc.

```
border-width: 5px;
```

border-style

This property is used to specify the style of the border. This property contains the following values,

```
solid -> _______
dotted -> ......
dashed -> -------
double -> ========
```

```
border-style: solid;
```

border-color

This property is used to specify the color of the border, here we can any CSS color values to apply a border color like textual color, RGB, HEX, RGBA, HSLA values.

```
Ex:
   border-color: black;
```

Shortcut property

In order to provide individual values, we can combine all the border styles using this shortcut property.

We can specify all the border styles using a shortcut property called 'border'

```
Syntax:
      border: width style color;
Example:
      boder: 5px solid green;
```

The above property will apply the border all the sides. We can even apply the border individual sides also using the following properties.

```
border-top: 60px solid lightsalmon;
border-right: 80px solid black;
border-bottom: 60px solid lightsalmon;
border-left: 80px solid black;
```

We can apply rounded corners for a border of a HTML Element is by using a CSS property called ‘ border-radius’.

```
border-radius: 20px;
```

From CSSS3 onwards we can even apply the individual sides of a border radius also. For this we have to use the following properties.

```
border-top-left-radius: 20px;
border-top-right-radius: 60px;
border-bottom-left-radius: 60px;
border-bottom-right-radius: 20px;
```

ROUNDED TWO

CSS Margins

The CSS Margins indicates some space around the border of any HTML Element.

Any white space which is around the border of any HTML element is comes under margin.

We can apply the individual sides of margin using the following CSS margin properties.

```
margin-top -> to apply the top margin for an element.
margin-right -> to apply the right margin for an element.
margin-bottom -> to applu the bottom margin for an element.
margin-left -> to apply the left margin for an element.
```

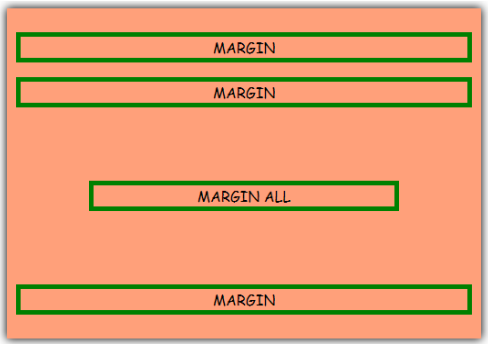
We can specify the margin in px , % , inherit values.

By using a shortcut property we can generate some margin for individual sides using a CSS property called ' **margin.**

```
margin: 5px 10px 15px 20px;
         T    R    B    L
margin: 5px 10px;
         TB  LR
margin: 5px;
        (All the sides)
```

The margin property values starts from top to left in a clock wise. If we provide two values for a margin property, then the first value indicates top and bottom margin. And the second value indicates the left and right side margin.

If we provide only value, this indicates the margin of all the sides.



CSS Paddings

The CSS paddings indicates some space inside the border and around the HTML Element.

By using CSS properties to generate some space for individual sides of any HTML Element.

```
padding-top -> to apply the top padding for an element.
padding-right -> to apply the right padding for an element.
padding-bottom -> to apply the bottom padding for an element.
padding-left -> to apply the left padding for an element.
```

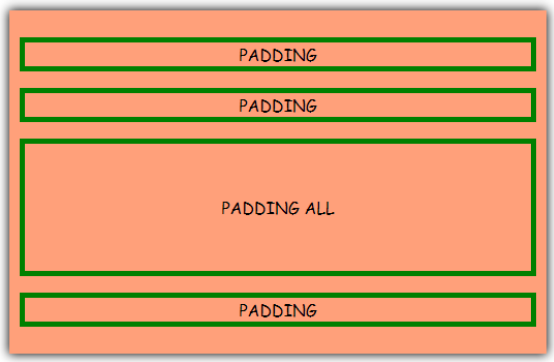
We can specify the padding in px , % , inherit values.

By using a shortcut property we can generate some space for individual sides using a CSS property called ‘ **padding’.**

```
Ex: padding: 5px 10px 15px 20px;
              T    R    B    L
   padding: 5px 10px;
             TB  LR
   padding: 5px;
            (all the sides)
```

The padding property values starts from top to left in a clock wise. If we provide two values for a padding property, then the first value indicates top and bottom padding. And the second value indicates the left and right side padding.

If we provide only value, this indicates the padding of all the sides.



CSS Outline

Usually CSS border are used to apply various border styles for any HTML Elements.
CSS Outline indicates same like a border, but which will be displyed outside the border.
In CSS outline is having the following properties same like border properties.

outline-width

This property indicates the width of an outline. We can specify the width of an outline in px, cm, mm, em, inch etc.

```
outline-width: 5px;
```

outline-style

This property is used to specify the style of the ouline. This property contains the following values,

```
outline-style: solid;
```

outline-color

This property is used to specify the color of an outline, here we can any CSS color values to apply a outline color like textual color, RGB, HEX, RGBA, HSLA values.

```
outline-color: red;
```

Shortcut property

In order to provide individual values, we can combine all the outline styles using this shortcut property.

We can specify all the outline styles using a shortcut property called 'outline'

```
outline: 20px solid orangered;
```

Note: In CSS There are no properties available for CSS outline properties like CSS Borders.

CSS Text Formatting

The CSS Text formatting properties are used to apply various styles to the HTML content which is displayed in the webpage. In HTML we use few tags for text formatting like <b> , <i> , <mark> tag etc. Using CSS we can apply styles for HTML content using various CSS properties. Those are as follows,

Color:

This is used to apply the color of the text on the webpage using any CSS color values like RGB , HEX , HSL , RGBA , HSLA , textual color values.

```
<div>
    <p style="color: red">Lorem ipsum dolor</p>
</div>
```

Lorem ipsum dolor sit amet, consectetur adipisicing elit. Adipisci beatae commodi cumque, doloremque et libero non! Accusantium aliquam amet animi aperiam asperiores aspernatur assumenda aut culpa cupiditate delectus dolore doloremque doloribus ea earum, enim et fugiat fugit hic in ipsum laboriosam libero mollitia nam nemo neque, nihil odio odit, placeat praesentium quasi quia quos rem reprehenderit sit unde voluptas voluptatem. Ad, assumenda autem consequatur dolores earum modi obcaecati officiis omnis, quisquam quos rerum velit vero! Ipsum, laudantium, voluptatem? Distinctio, libero.

text-align

This is used to align the text to either left or center or right side of the browser.

```
<div>
    <h1 style="text-align: center">I am Center Aligned</h1>
</div>
```

I am Center Aligned

text-decoration

This is used to apply the decoration effect for the HTML content.
This property contains the values like underline , overline , none.
We normally use this property to remove the underline for the <a> tags.

Ex: text-decoration:none;

```
<div>
    <h1><a href="#" style="text-decoration: none">I am an Anchor Tag</a></h1>
</div>
```

I am an Anchor Tag

text-transform

This is used to convert the text to lowercase or uppercase or Capitalize.

```
<div>
    <p style="text-transform: uppercase">Lorem ipsum dolor sit...</p>
</div>
```

LOREM IPSUM DOLOR SIT AMET, CONSECTETUR ADIPISICING ELIT. DESERUNT DUCIMUS EOS ERROR ESSE ET ITAQUE PRAESENTIUM PROVIDENT SOLUTA ULLAM. A ADIPISCI ALIQUAM CONSEQUATUR FUGIAT LABORE, LABORUM MINIMA MINUS ODIT OFFICIA PORRO RATIONE SIT VERO VOLUPTAS. A ACCUSANTIUM ALIQUAM ALIQUID ASPERNATUR CONSECTETUR DEBITIS DOLOR EIUS ELIGENDI, ENIM EST EX FACILIS MAGNAM NAM NEQUE NIHIL NON NULLA ODIO ODIT OFFICIIS PARIATUR PRAESENTIUM QUO QUOS RERUM, SUNT TEMPORA UT VOLUPTAS. ALIQUAM ANIMI DOLORES ERROR HIC IPSUM MINIMA MOLESTIAS MOLLITIA QUI, RECUSANDAE RERUM? NEQUE?

text-indent

This property is used to indent some space for the starting of the first line of a paragraph.

```
<div>
    <p style="text-indent: 80px">Lorem ipsum dolor sit...<p>
</div>
```

Lorem ipsum dolor sit amet, consectetur adipisicing elit. Aperiam architecto beatae consequatur corporis eos esse id ipsam maiores neque nesciunt. Alias amet dignissimos dolor ipsum labore natus possimus quaerat quis sapiente, similique sint vel voluptates voluptatum. Aliquid aperiam asperiores aut consectetur consequuntur cupiditate dolore dolores ducimus esse et expedita, facilis hic illum impedit itaque libero nesciunt nostrum perferendis possimus quas quos reprehenderit temporibus veniam veritatis voluptas voluptatem voluptates voluptatibus. Esse itaque iure laborum nesciunt officiis quas ratione unde vero voluptatum?

letter-spacing

This is used to apply the custom space for each letter of a paragraph.

```
<div>
    <p style="letter-spacing: 10px">Lorem ipsum dolor sit...</p>
</div>
```

Lorem ipsum dolor sit amet, consectetur adipisicing elit. Accusantium ad aut cupiditate deleniti dolore dolorem dolores eaque eius error, facilis fuga fugiat impedit ipsam iusto maxime minus molestias, mollitia neque nisi odio officia optio possimus quaerat, quisquam ratione repellat reprehenderit saepe soluta tempore voluptas? Dolores esse hic nemo soluta ullam? Aliquid consectetur eum magnam maiores necessitatibus pariatur perferendis quisquam tempore totam voluptatibus. Amet dolor incidunt ipsa labore laudantium maxime, nobis nostrum perspiciatis quasi repellat repellendus voluptates. Commodi dicta fugit unde?

line-height

This is used to apply the custom line height for each line of a paragraph.

```
<div>
    <p style="line-height: 15px">Lorem ipsum dolor sit...</p>
</div>
```

Lorem ipsum dolor sit amet, consectetur adipisicing elit. Animi cumque dicta laboriosam natus neque porro provident reprehenderit sint veritatis, vero. Aliquam at, atque delectus eveniet exercitationem ipsam ipsum molestias perspiciatis, porro reiciendis sequi sit unde voluptatibus. Corporis delectus eum fugit id illum ipsam necessitatibus nihil qui ullam ut, Adipisci alias assumenda aut beatae consequuntur distinctio doloremque dolorum earum est, exercitationem facere incidunt iure laborum, laudantium nemo nisi odio officia perferendis quod repellat sapiente sint tempore temporibus tenetur ut velit veniam!

Direction

This is used to change the direction of the text to LTR or RTL.

```
<div>
    <p style="direction: rtl">Lorem ipsum dolor sit...</p>
</div>
```

Lorem ipsum dolor sit amet, consectetur adipisicing elit. Consectetur deleniti sequi vel! Ad consectetur ex excepturi id quam tempora. Commodi corporis doloremque facere facilis laborum modi molestias necessitatibus non, quidem, quis reiciendis repudiandae veniam voluptatum. Aliquid cumque delectus eaque et eveniet facilis libero minima, obcaecati odio reprehenderit velit voluptatibus. Alias blanditiis dignissimos ducimus est ex fugit illum impedit, iste laboriosam libero maiores nam odit repudiandae vel voluptas voluptates, .voluptatum. Commodi debitis odit quod similique vero? Impedit molestiae nulla perferendis quisquam

word-spacing

This is used to apply the custom space for each word of a paragraph.

```
<div>
    <p style="word-spacing: 15px">Lorem ipsum dolor sit...</p>
</div>
```

Lorem ipsum dolor sit amet, consectetur adipisicing elit. Accusamus, adipisci aspernatur assumenda autem consequuntur cum debitis deleniti dignissimos dolor eaque earum eveniet explicabo facilis fugit hic in ipsam labore laboriosam laborum necessitatibus non odio quae quam quidem quo quos, ratione suscipit tempore tenetur totam unde ut veritatis vero. Ad aliquid animi, aperiam at autem beatae commodi dolorem eaque enim eos et explicabo hic in incidunt maxime minima molestias, nemo nostrum pariatur placeat quaerat quis repellat repellendus repudiandae similique tempore vero?

text-shadow

This is used to apply the shadow effect for any HTML content.

```
Ex: text-shadow: 5px 5px red;
                (X-axis,Y-axis,color)

<div>
    <h1 id="text-shadow">UiBrains.com</h1>
</div>

#text-shadow{
    text-align: center;
    font-size: 80px;
    text-shadow: 5px 5px yellow;
}
```

UiBrains.com

CSS Fonts

The CSS Fonts are used to apply the font styles for the HTML Content like font-family , font-size , font-weight and font-styles.

The CSS Fonts contains the following properties

font-family

This property is used to apply the type of the font to be applied for the HTML content.

```
Syntax:
   font-family:font-name,"family";
```

In CSS all the fonts are categorized into four families, such as

1) serif (Times New Roman)

2) sans-serif (Arial , Verdana, Georgia)

3) monospace (Lucida Console)

4) cursive (most of the Google fonts)

```
#arial{
    font-family: Arial, sans-serif;
}
```

I am From Arial Font Local

```
#google{
    font-family: 'Indie Flower', cursive;
}
```

I am From Some Font Google

font-style

This property is used to convert the text from normal to Italic.

```
#font{
    font-style: italic;
}
```

font-weight

This property is used to convert the normal text to bold or bolder.

```
#font{
    font-weight:bold;
}
```

font-size

This is used to apply the custom font size for the HTML element in px , cm , mm etc.

```
#font{
    font-size: 14px;
}
```

```
#custom{
    font-family: 'Indie Flower', cursive;
    font-size: 50px;
    font-weight: bold;
    font-style: italic;
}
```

I am a Custom font

CSS Links

CSS are used to apply the custom styles for hyperlinks for some special events using CSS pseudo selectors.

Usually each HTML hyperlinks are having the default styles like un-touched links with blue in color and underline and visited links will in purple color. We can customize these styles using CSS links.

The CSS links pseudo selectors are as follows,

a:link

This pseudo selector indicates a normal unvisited and un-touched link.

a:hover

This pseudo selector indicates the moment a user mouse hover on a hyperlink.

a:active

This pseudo selector indicates the moment a user click on the link and but not released the cursor on the link.

a:visited

This pseudo selector indicates a link which has been visited already.

By using these CSS links properties we can customize the styles for these events.

UiBrains.com

```
a:link{
    background-color: forestgreen;
}
a:hover{
    background-color: orangered;
}
a:active{
    background-color: black;
}
a:visited{
    background-color: blue;
}
```

Google

CSS Tables

Using CSS styles properties we apply some more styles for a HTML table and convert like a fancy table as follows,

To select even number of rows and odd number of rows we use the following pseudo selectors.

```
tr:nth-child(even){
    background-color: lightsalmon;
    color: black;
}

tr:nth-child(odd){
    background-color: lightcoral;
    color: black;
}
```

Emp Name	Designation	Company
John Mayor	Software Engineer	Microsoft
Alex Ben	Software Engineer	Google
John	Software Engineer	Microsoft
Alex Ben	Software Engineer	Google
John	Software Engineer	Microsoft
Alex Ben	Software Engineer	Google

CSS Positions

The CSS Positions are used to place the HTML Elements in the specified positions on the webpage.

For these styles we use a CSS property called ‘position’.

This CSS position property contains the following values,

position:static

By default all the HTML elements are defined with the position ‘static’. If we mention them specifically or not by default all the HTML elements are with the default position ‘static’. If any HTML element is defined with static position, then its movement is not effected by top, left, bottom and right properties.

It means any HTML element with static position, then its position is always static and no change in the movement of it.

position:relative

This is opposite to the static position. If any HTML element is defined with this position then its movement is effected by top, left, bottom, right properties.

A HTML Element with this position, its movement is independent and it’s not depend on any other HTML element. We normally use this position with CSS animation to move a HTML element in the specified places on the webpage.

position:absolute

This is almost same as relative position, if any HTML element is defined with this position then its movement is also effected by top, left, bottom, right properties.

But any HTML elements is defined with this position then its movement is purely depend on the enclosed parent element or ancestor element.

If the ancestor element is moving then this element with this position also moves along with the patent.

We normally use this to stick the dropdown list with the parent element.

UiBrains.com

position: fixed

Any HTML element is defined with this position, its movement is also effected by top, left, bottom, right properties but once moved to specified position and its fixed in that position. It means with respective to scroll bar, this is not effected with it and it's always fixed in the specified position.

We normally use this position to fix the main navigation bar at the top of the webpage or to fix the main footer at the bottom of the webpage.

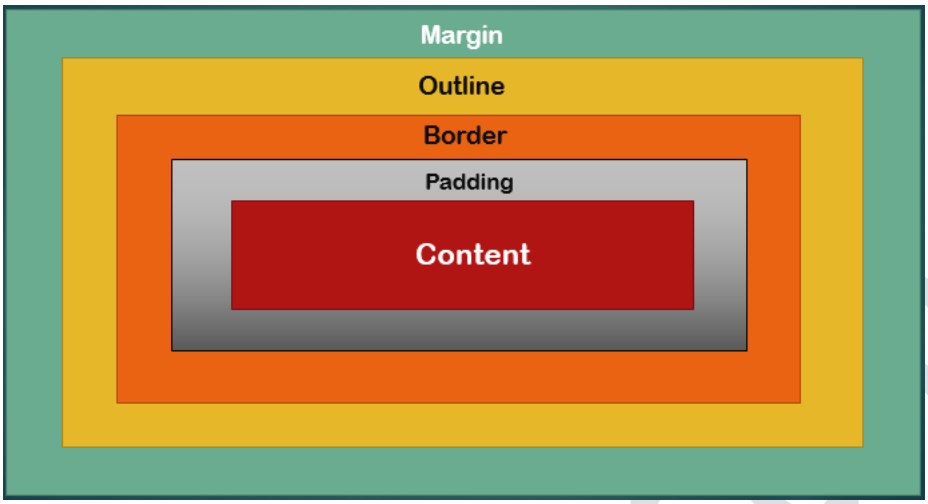
position:sticky

This position is quite different than all other CSS position. Any HTML element is defined with this position then its movement is purely depend on the scroll bar.

We normally use this to display an advertisement box on the webpage.

uBrains.com

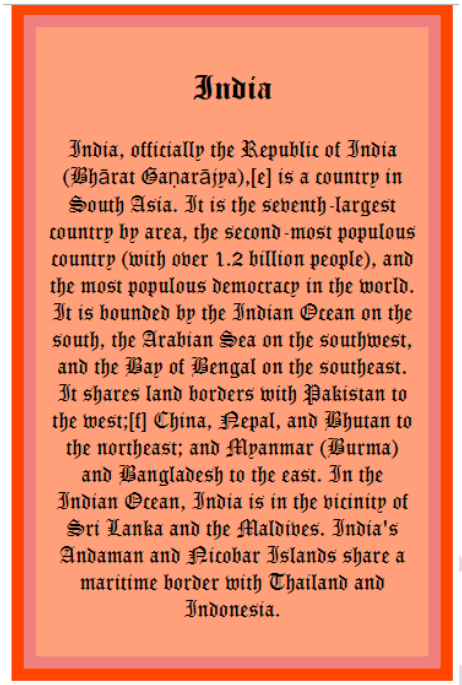
CSS Box Model



The CSS Box model consists of all CSS layout properties like padding, border, outline, margin properties. Once we apply all the above properties the HTML Content will looks like a box layout.

We can apply all these properties for a HTML Element and display like a box model on the webpage.

```
#india{
    border: 10px solid lightcoral;
    width: 300px;
    text-align: center;
    font-family: "Old English Text MT",sans-serif;
    font-size: 18px;
    padding: 10px;
    background-color: lightsalmon;
    outline: 10px solid orangered;
    margin: auto; /* to make in the center */
}
```



CSS Display Property

Background Information

The HTML Elements are used to display a content on a webpage. The content which is displayed on a webpage is not based on how we write the HTML tags in a HTML document. The content which is displayed on a webpage is purely depend on the actual display nature of the specific HTML element.

Each HTML element / Tags are having its own default display nature.

In HTML, the elements / tags are divided into two categories based on their display nature. They are as follows,

```
1) Inline Elements
              <a> , <span> , <input> , <button> , <img>

2) Block Level Elements
            <p> , <div> , <h1>...<h6> , all HTML5 layout tags
```

The HTML tags like the above mentioned are comes under inline elements. For these tags even though we write them line by line or in a single line in a HTML document, then they will be displayed in a same line on the webpage. These tags are called as Inline Elements.

The HTML tags like the above mentioned are comes under Block level elements. For these tags even though we write them line by line or in a single line, they will display line by line only on as webpage. These elements are called as block level elements.

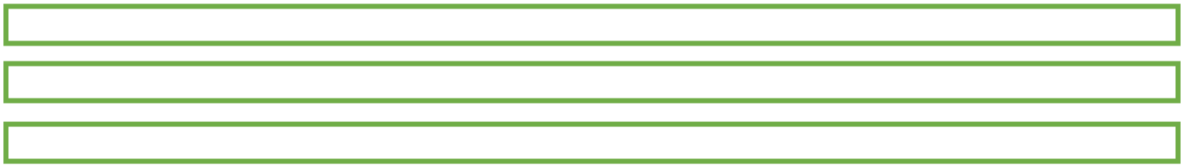
The display of HTML elements on a webpage is purely depend on its default display nature. By using CSS 'display' property we can change the default display nature of HTML elements like from Block level elements to inline and inline elements to block level.

The CSS display property contains the following values.

display:inline

This property value is used to convert the block level elements to inline.

display:block



This property value is used to convert the inline elements to block level.

display:inline-block

This property value is used to display HTML elements in the same line as a blocks. We normally use this to display hyperlinks on a main navigation bar.

display:none

This property value is used to hide any HTML element to display on the webpage.

We normally use this to hide any dropdown list, we will make it display block while we hover on the dropdown button.

CSS float property

The CSS float property is used to float any HTML element to either left or right side of the webpage.

Normally all the block level HTML Elements will occupy the full width of the browser and they will not let the next elements to display aside to them.

For this we will apply the float effect for the first HTML Element to keep it either left or right side of the webpage.

In CSS Floats we use a property called 'float' and this contains two values such as

```
float:left;
float:right;
```

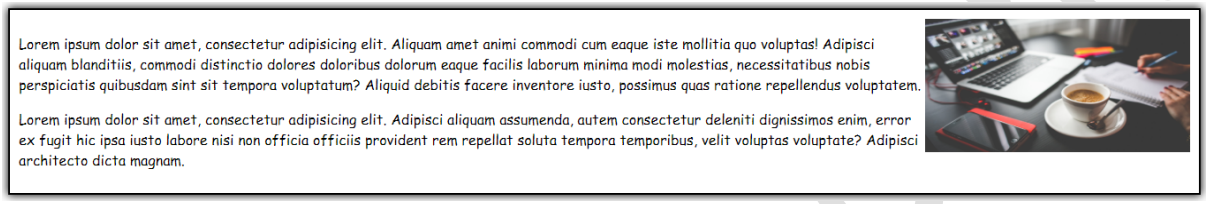
This 'float' property may disturbs the display of the next elements.

In order to clear the float effect for the next HTML elements, we use a property called 'clear'

```
clear:both;
```

Note: We normally use this property to display a 'sidebar' on either side of the webpage.

We also use this to display the newspaper article to surround the content with images.



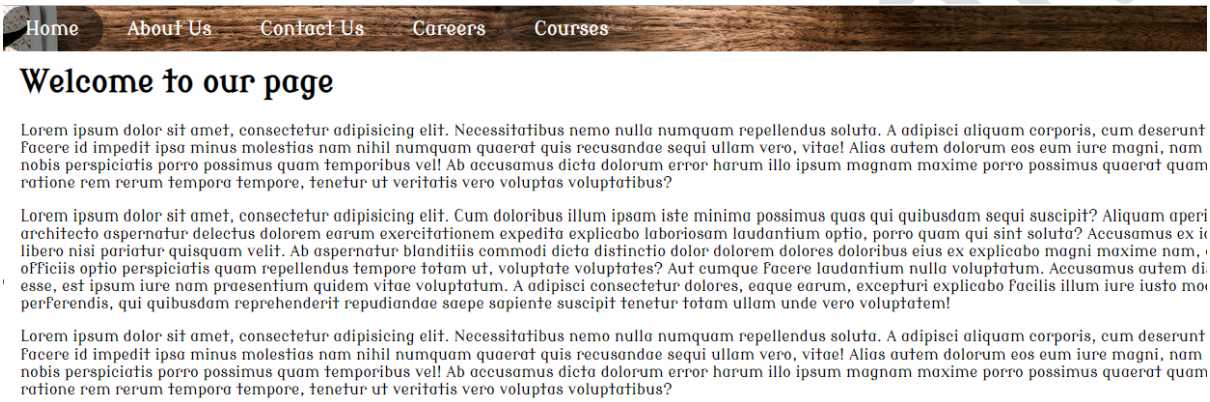
CSS Vertical Navigation Bar

By using the above properties we can display a vertical navigation bar as follows,

```
<aside>
    <ul>
        <li>
            <a href="#" class="active">HTML</a>
        </li>
        <li>
            <a href="#">CSS</a>
        </li>
    ........
    </ul>
</aside>
```

Horizontal Navigation Bar

```
<nav>
    <ul>
        <li>
            <a href="#" class="active">Home</a>
        </li>
        <li>
            <a href="#">About Us</a>
        </li>
        <li>
            <a href="#">Contact Us</a>
        </li>
    ......
    </ul>
</nav>
```

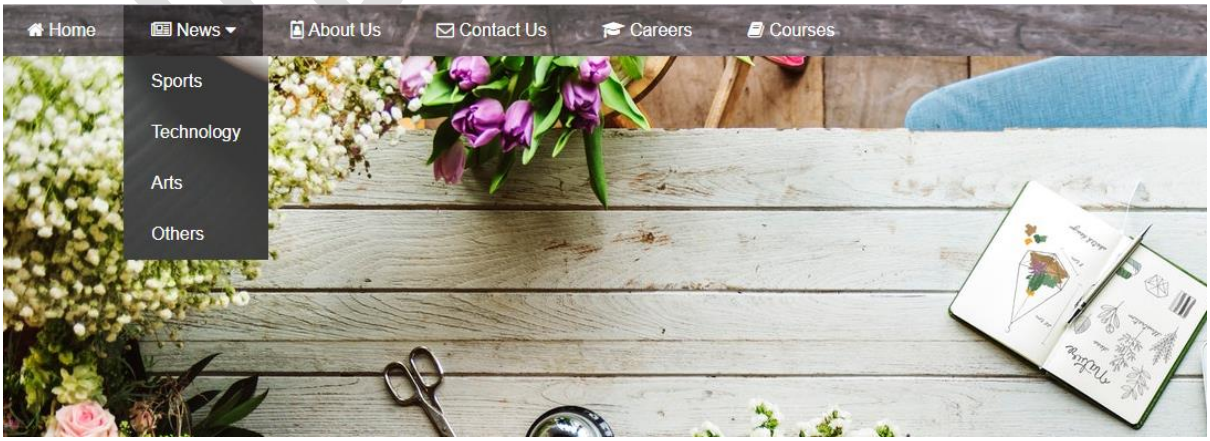


CSS Dropdown Menu

```
<div class="dropdown">
    <button class="dropdown_button">Courses
        <i class="fa fa-arrow-circle-o-down"></i>
    </button>
    <div class="dropdown_list">
        <a href="#">HTML</a>
        <a href="#">CSS</a>
        <a href="#">JavaScript</a>
        <a href="#">JQuery</a>
        <a href="#">AJAX</a>
        <a href="#">Bootstrap</a>
        <a href="#">Angular JS</a>
    </div>
</div>
```

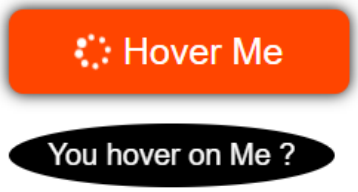
CSS Dropdown Navbar

```
<nav>
    <ul>
        <li>
            <a href="#">Home</a>
        </li>
        <li class="dropdown">
            <a href="#">News</a>
            <div class="dropdown_list">
                <a href="#">Sports</a>
                <a href="#">Technology</a>
                <a href="#">Arts</a>
                <a href="#">Others</a>
            </div>
        </li>
    ........
    </ul>
</nav>
```



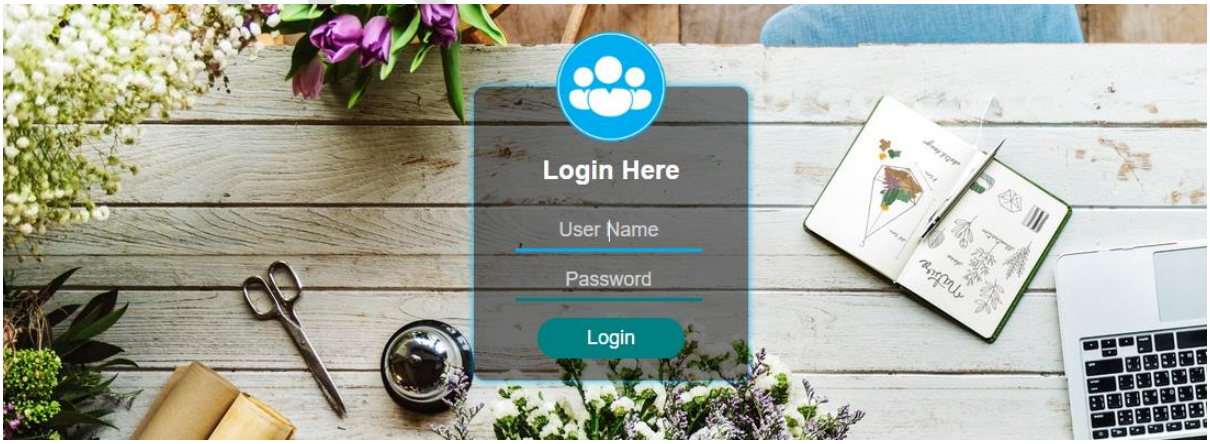
CSS Tooltip

```
<div class="tooltip">
    <button class="tooltip_btn">Hover Me</button>
    <p class="tooltip_text">You hover on Me ?</p>
</div>
```



CSS Forms

```
<!-- Login Form -->
<div class="form-div">
    <img src="img/user.png">
    <h2>Login Here</h2>
    <form>
        <input type="text" placeholder="User Name">
        <br>
        <input type="password" placeholder="Password">
        <br>
        <input type="submit" value="Login">
    </form>
</div>
```



CSS Image Gallery

```
<h1>Image Gallery</h1>

<div class="image">
    <img src="img/image4.jpeg" width="600" height="400">
    <div class="desc">Image 1</div>
</div>

<div class="image">
    <img src="img/image6.jpeg" width="600" height="400">
    <div class="desc">Image 2</div>
</div>
```

CSS Transformation

The Transformation is an effect that lets an element to change its shape, size and position

For CSS Transformation we use the property called 'transform'

The transform property allows us to visually manipulate an element by skewing, rotating, Translating or Scaling.

CSS3 Supports 2D and 3D Transforms

In CSS 2D Transformation it contains the following methods for 'transform' property.

translateX()

The translateX() method moves an element along X-axis by the given parameters.

```
Ex: transform: translateX(10px);
```

translateY()

The translateY() method moves an element along Y-axis by the given parameters.

```
Ex: transform: translateY(10px);
```

translate()

The translate() method moves an element along X-axis and Y-axis by the given parameters.

```
Ex: transform: translate(10px,20px);
```

scaleX()

The scaleX() method is used to increase or decrease the size of an element using the provided parameters.

```
Ex: transform:scaleX(2); (Increases width 2 times)
```

scaleY()

The scaleY() method is used to increase or decrease the size of an element along Y-axis.

```
Ex: transform:scaleY(2); (Increases height 2 times)
```

scale()

the scale() method is used to increase or decrease the size of an element along both X-axis and Y-axis.

```
Ex: transform:scale(2,3);
```

```
scale value    effect
-----------    --------
   <1           decreases
   >1           Increases
   =1           No Change
```

skewX()

The skewX() method is used to skew an element along the X-axis by the given parameters.

```
Ex: transform:skewX(45deg);
```

skewY()

The skewY() method is used to skews an element along the Y-axis by the given parameters.

```
Ex: transform:skewY(45deg);
```

skew()

The skew() method is used to skew an element along the X-axis and Y-axis by the given parameters.

```
Ex: transform:skew(45deg,45deg);
```

matrix()

This method is a shortcut for all the above 2D transform methods.

In this method we will combine all the 2D transform methods using a mathematical funciton.

```
Syntax :

transform: matrix(scaleX(),skewY(),skewX(),scaleY(),translateX(),translateY());
```

rotate()

The rotate() method is used to rotate an element in clock wise or counter clockwise based on the given parameters.

If we provide the +ve degrees it rotates in the clockwise , if we provide the -ve degrees it rotates in anti-clockwise.

```
Ex: transform:rotate(45deg);
```

CSS 3D Transformation

3D Transform allows the elements to transform in 3D way.

CSS 3D transformation contains the following methods as values for a 'transform' property.

rotateX()

The rotateX() method rotates an element around its X-axis at a given degree

```
.rotatex:hover{
    transform: rotateX(180deg);
}
```

rotateY()

The rotateY() method rotates an element around its Y-axis at a given degree

```
.rotatey:hover{
    transform: rotateY(180deg);
}
```

rotateZ()

The rotateZ() method rotates an element around its Z-axis at a given degree

```
.rotatez:hover{
    transform: rotateZ(180deg);
}
```

CSS3 Transitions

CSS3 transitions allows us to change property values smoothly (from one value to another), over a given duration.

To create a transition effect, we must specify two things, such as

The CSS property we wants to add an effect, the duration of the effect

For this CSS transition we use a property called 'transition'. The CSS Transition following properties.

transition-name -> indicates the name of the transition

transition-duration -> indicates the duration of the transition in seconds or milliSeconds.

transition-delay -> to delay the transition in the specified number of seconds or milliSeconds.

transition-timing-function -> indicates the timing-function of the transition effect.

-> This property contains the following values,

```
linear -> slow start , slow end
ease-in -> slow start , fast end
ease-out -> fast start , slow end
ease-in-out -> fast start , fast end
```

CSS3 Animations

An animation lets an element gradually change from one style to another.

CSS3 animations allows animation of most HTML elements without using JavaScript or Flash!

You can change as many CSS properties you want, as many times you want.

To use CSS3 animation, you must first specify some key frames for the animation.

Key frames hold what styles the element will have at certain times.

The CSS3 Animations contains the foollowing properties

animation-name : indicates the name of the animation

animation-duration : indicates the duration of an animation in seconds or milliseconds.

animation-direction : indicates the direction of an animation.

animation-iteration-count : indicates the number of times an animation to be repeated.

animation-delay : indicates the delay time frame of an animation in seconds or milliseconds.

animation-timing-function : indicates the timing function of an animation.
The values of this property are exactly same as transition-timing-function.

```
.simple{
     width: 100px;
     height: 100px;
     background-color: orangered;
     border-radius: 50%;
     border: 20px solid orange;
     box-shadow: 0 0 15px black;
     animation: simple 2s infinite linear;
 }

@keyframes simple {
    0%,50%,100%{
        background-color: orange;
        border-color: orangered;
    }
    25%,75%{
        background-color: orangered;
        border-color: orange;
    }
}
```

Simple Animation

```
.spin{
    width: 100px;
    height: 100px;
    border-radius: 50%;
    border: 20px solid orange;
    border-top: 20px solid darkred;
    border-bottom: 20px solid darkblue;
    animation: spin 2s infinite linear;
}
@keyframes spin {
    0%{transform: rotate(0deg)}
    100%{transform: rotate(360deg)}
}
```

Spin Animation

News Animation

Latest News: Lorem ipsum dolor sit amet, consectetur adipisicing elit. Iure, voluptatibus

Box Animation

Foot Ball Animation

CSS Buttons

```
<h2>Simple Buttons</h2>
<div>
    <a href="#" class="btn">Link</a>
    <button class="btn">Button</button>
    <input class="btn" type="button" value="Input">
</div>
```

Simple Buttons

Link Button Input

```
<h2>Simple Outline Buttons</h2>
<div>
    <a href="#" class="btn-outline">Link</a>
    <button class="btn-outline">Button</button>
    <input class="btn-outline" type="button" value="Input">
</div>
```

Simple Outline Buttons

Link Button Input

```
<h2>Bootstrap Buttons</h2>
<div>
    <button class="bs-btn primary">Primary</button>
    <button class="bs-btn secondary">Secondary</button>
    <button class="bs-btn success">Success</button>
    <button class="bs-btn warning">Warning</button>
    <button class="bs-btn danger">Danger</button>
</div>
```

Bootstrap Buttons

Primary Secondary Success Warning Danger

```
<h2>Bootstrap Outline Buttons</h2>
<div>
    <button class="bs-btn-outline outline-primary">Primary</button>
    <button class="bs-btn-outline outline-secondary">Secondary</button>
    <button class="bs-btn-outline outline-success">Success</button>
    <button class="bs-btn-outline outline-warning">Warning</button>
    <button class="bs-btn-outline outline-danger">Danger</button>
</div>
```

Bootstrap Outline Buttons

Primary Secondary Success Warning Danger

JavaScript

What is JavaScript?

JavaScript is a lightweight, interpreted, object oriented language with first class functions and it is more popular as a scripting language for webpages and also used in many non-browser environments.

Lightweight

JavaScript does not require any large kind of setup like any other languages like java.

In order to execute a simple java program, we need to do the following,

Download and install JDK, setting class path, writing a java program, compile it and run it.

When comes to JavaScript we can directly write a JavaScript in a file and execute it directly without any compilation process, and even JavaScript occupies small amount of memory an easy to implement it. Because of this we call JavaScript is a light weight programming language.

Interpreted Language

In JavaScript the program will be executed directly which does not require any compilation process and no generation of intermediate files to execute the java program.

As Java and C++ are compiled and Interpreted programming languages, means we needs to write a java program and compiles to a class file and that has to be executed.

When it comes to JavaScript, the instructions will be executed directly.

Note:

Technically there is a compilation process happens which Is totally different than any other object oriented language. We will discuss about this compilation concept JavaScript in the later sections.

UiBrains.com

Object Oriented

JavaScript is an object oriented programming language.

An Object orientation is a concept where in order to solve any problem, instead of writing some set of instructions to be executed, we actually provide a solution by breaking down the problem into the real world objects.

JavaScript is also an object oriented programming language which uses real world objects to solve any problems such as any other object oriented programming language does.

First Class Functions

This is one of the concept of functional programming wherein we can use functions as values and also we can use functions as an arguments to any other methods just like values of a variable.

For example we can assign a string values to a variable as

```
var name = "Naveen";
```

We can also assign a object to a variable as

```
var employee = new Employee();
```

The same w can use a function itself as a value and assign it to a variable.

Note:

It clearly state that we are not going to execute a function and assigning the return value to a variable, instead we will assign the whole function itself to a variable.

We can also use functions itself as a parameters to any other function.

This is one of the major feature of functional programming language.

As JavaScript is also actually originated from the family of 'C' Language which is having this feature already in it.

Scripting Language

A scripting language is basically a language where set of instructions are written for a runtime environment.

The best example for scripting language is "shell" scripting.

UiBrains.com

In the Linux or UNIX platform we can execute various commands targeted to any runtime environments like 'listing' a directory and copying files from one place to other.

Instead of executing each command, we can actually write those commands one by one and bundled into a script file called .sh file. And that file will be executed with UNIX or Linux runtime.

These instructions / commands will be executed by Linux server directly.

The same way in JavaScript we will be writing a set of instructions and make it to a '.js' file and this will be executed by the target runtime typically a web browser.

Due to this JavaScript is called as scripting language.

There are different runtime environments for JavaScript along with the web browser is a server.

We can even execute a JavaScript in server like environments.

Ex: Node JS, Express JS

Note: Node JS is a native JavaScript which is used to execute the JavaScript in the server environments.

Express Js is a framework built on top of node js and also used for server side scripting.

Both Node JS and express JS are server side scripting languages, whereas native JavaScript, JQuery are browser side scripting languages.

Weakly Typed Language

A Weakly typed language is a language where the strict type checking is not performed before executing a program.

Basically the 'Java' language is a strongly typed language were the type checking of each and every variable and proper syntaxes before the compilation of a program.

JavaScript is actually the most forgiving language and suits to the basic level of programmers in mind.

Whereas java is actually meant for professional programmers where they need to know each of the rule of the language.

UiBrains.com

Strictly Type Language

Basically java, C++ and other programming languages are strictly typed or strongly typed languages where each and every variable method, class are strictly checked by the compiler.

There are actually a lot of strict rules to be followed in order to write and compile a program in java.

Where in JavaScript this strictly typing is not required, and it is a weakly typed language.

Note: due to this weakly typed feature of JavaScript there are some wired design designs are made in order to provide backward compatibility of the language.

Understanding Runtime Environment for JavaScript

A Typical runtime environment for JavaScript a browser.

Browser send on http requires to a server and server send back an http response to the browser.

This http response from the server is actually bunch of text which contains a set of HTML tags with data.

Browser gets the HTML code and builds a tree like structure called a DOM Tree.

Diagram:

After building a DOM Tree, it renders the page and display the rendered page on the browser.

Whenever we load a html on the browser, we get some sort of view on the browser.

If we load the same html again and again we still get the same vie and which will not be changed.

As the HTML is a static language, whenever we loads and any number of times the view will be same.

When the browser loads html code along with JavaScript, the view may get change.

As JavaScript is a dynamic programming language is having an ability to change the DOM tree of a browser.

By using JavaScript we can

1. Add a new node to a tree
2. Remove a node from a tree.

UiBrains.com

3. Edit a node from a tree

Because of all these modifications, browser builds a new Dom tree or refactor the DOM tree and render the page view.

As the browser refactor the DOM tree, obviously we will get another view and static view as with the HTML.

Why Learn JavaScript

By using JavaScript we can develop a rich client side applications.

In any web application the client side web development can be done using JavaScript.

There are some other areas of usage of JavaScript as follows,

1. Client Side Applications.
2. Server side Applications.
3. Browser extensions.
4. Desktop applications
5. Mobile Applications
6. IOT Applications.

Client Side Applications

As JavaScript is most popular in client side web development.

JavaScript is the world most popular Browser's we scripting language.as of today.

By learning JavaScript we can develop a rich client side applications.

There are some other frameworks or libraries we can use to develop client applications along with native JavaScript.

Once we learn JavaScript, It will be easy for us to learn and understand the following technologies such as JQuery, Angular Js React JS Ember and Backbone JS.

As Angular is nowadays most polar JavaScript frame work to build rich client side applications and SPA (Single Page Application).

With JavaScript knowledge we can be able to learn and understand AngularJS better way.

UiBrains.com

Server side Development

JavaScript also gained popularity in server side environments as well.

Ex: node js, Express JS.

By using Node JS and Express JS we can develop server side logic development. Express JS is a Library of Node JS.

Browser Extensions:

There are lot of browser extensions available in the market are actually built using JavaScript.

Example:

Add Blocker for Chrome

Eye Dropper.

Desktop Applications:

By using JavaScript we can even develop some of the desktop applications using JavaScript.

Example: Web Torrent Desktop

WordPress Desktop Application

Pexels Desktop Application.

We can use some of the tools like 'electron' which is a software development platform created by GitHub that lets developers to use JavaScript along with HTML, CSS to create Desktop applications..

Mobile Applications:

By using JS we can even develop some of the mobile applications using JavaScript and other tools.

Popular tools for developing mobile apps using JS are

Phone GAP / Cordova, Titanium / JQuery Mobile / Meteor

UiBrains.com

IOT Apps

JavaScript is the Top programming language for building IOT Apps.

IOT stands for Internet of Things.

IOT Is the network of physical devices, vehicles and other items embedded with electronics, software sensors with network connectivity to collect and exchange the data.

Example:

By using IOT apps we can control our home things like washing Machine, TV or A/C from anywhere in the world.

Interesting Thoughts of JavaScript:

Duglas Crockford is one of the famous person in JavaScript landscape stated that

JavaScript is the world's most misunderstood programming language

There is actually a reason behind this statement as there are some unique features of JavaScript misunderstanding concepts which leads the developer gets frustrated because we would not expect the things works in JavaScript and some of the unexpected behavior of JavaScript.

Example: This keyword, __proto__

Actually this is one survey happened in the early days of JavaScript on the concept of

The most loved language and the most hated language

After the survey the results announced and surprisingly "JavaScript" is top most among the most loved language and the most hated language list.

Why the JavaScript became the most hated language list to most of the developers is that due to lack of understanding of actual concepts and behavior of JavaScript.

Actually People / Developers from other server side language like Java, may think like JavaScript is same as Java and behaves like Java.

Actually JavaScript is syntax wise same as Java, and but there are some concepts which are not behaves as we expected. And there also some of the features which are unique to JavaScript.

Ex: IIFE, This keyword

UiBrains.com

In this initial days of development we may feel we are aware of JavaScript but once we started writing some serious JavaScript code then the developer comes to know the importance of knowing concepts of JavaScript.

History of JavaScript:

JavaScript is created by Brendan Eich in Netscape during mid 90's while the browser war between internet explorer and Netscape navigator.

While Brendon was with Netscape, there Java which already popular in those days with the concept of Applets which provide rich client Interface those days.

Initially JS is named live script but people wanted to use the popularity of Java , Later renamed it to JavaScript.

JavaScript is just to complement Java, it is named as JavaScript and not having any relation with Java.

Brendon created JavaScript around span of 10 days and rushed to Production.

JS is developed new guys in mind and it is a weakly typed and most forgiving language.

Again due to the competition which Microsoft internet Explorer, JS people wanted to standardize it.

Once it is standardize it will become a specification and all the browsers has to follow the specification.

Js is standardized by a committee called "ECMA" and it called as ECMA Script.

There are various version of EXMA Script and the latest version of ECMA is ECMA 8.

But it is not fully supported by all the modern browsers. The only supported version as of today is ECMA 5.

We will be learning ES5 version of JavaScript.

Setting Up Development Environment:

For any language development environment is required. For JavaScript the runtime environment is a web browser.

Once we install any browser that is enough with the development environment.

Example:

Mozilla, Chrome, IE, Opera, Safari.

UiBrains.com

Here in this course we will be using Mozilla.

Steps:

Download and install Mozilla Firefox Browser.

Open the browser window

Click on Developer Tools and Scratchpad.

1) Toggle Tools -> Console Tab
2) Web Tools -> Console Tab
3) Scratch Pad

Almost all the browsers comes with an inbuilt JavaScript engine and some tools to write code using JavaScript.

Example:

```
1  console.log("Hello JavaScript");
```

Console -> Global Object

Log -> function of Global Object

Hello JavaScript -> the String to be printed on the console.

We can also use Scratch pad to write code.

Steps:

- ➔ Open a Scratchpad.
- ➔ Write JavaScript code.
- ➔ Right Click and click on Reload and Run
- ➔ Code will be executed by the browser.
- ➔ Logs will be printed on the console.

Variables and Declaration:

In all the other languages there all certain variable and which holds some values.

Any variables acts like a container and which holds some value of it.

The variable declaration JavaScript as follows,

Example:

UiBrains.com

```
var num1 = 10;
```

var -> keyword to declare a variable

num1 -> name of the variable

10 -> value of the variable num1.

Note:

In JavaScript we will not specify any type of variable like Java. Any variable can be declared using “var” keyword.

Example:

```
var name = "Naveen";
var isStudent = true;
```

JavaScript does not have a concept of variable types.

We can break declaration into two likes also.

```
var name ; // declaration
name = "Naveen"; // Definition
```

In JavaScript there is no scoping related keywords like public or private like JAVA.

Every variable which is declared is always a public in JavaScript.

JavaScript is having the following primitive types

1) Number
2) String
3) Boolean
4) Undefined
5) Null

Number:

Number is a JS Primitive type for storing numbers.

Number in JS are “Double Precision 64 bit format IEE 754 values”.

It means in JS we don’t have integers as such. All the Numbers which are declared are floating points Numbers only.

As in Java We might have byte, short, int, long, float double types for declaring numbers.

But in JS we have the only type called Number to declare any type of numbers.

Example:

```
var a = 10;
Console.log(a); // 10

var num1 = 10;
var num2 = 20;
var sum  = num1 + num2;
Console.log(sum); // 30
```

Note:

We can define Integer Values we can be able to see the integer values as result. But JavaScript Actually maintain them as a 64bit floating point numbers.

String:

String in JavaScript is a collection of characters.

String is a collection of 16 bit Unicode characters.

In JavaScript there no character datatype. And each character is just a string with the length of ONE.

Note:

In JavaScript any text values which is assigned to a variable is of type “String” only.

Example:

```
var name1 = "Naveen";
console.log(name1); // Naveen
var name2 = "Naveen";
console.log("Hello " + name2); // Hello Naveen
```

UiBrains.com

Boolean

Boolean is a JS primitive datatype to represent true / false values.

This Boolean datatype are can use in any conditional statements.

```
var isStudent  = true;
console.log(isStudent);
```

JS Typing Interchange

In JavaScript the variable type are interchange means, once we declare a variable pf pne type we can assign the value of another type.

Example:

```
var a = 10; // a is a Number
a = "Hello"; // a is String
a = true; // a is Boolean
```

In any other languages like Java, once a variable is declared of one type it cannot be interchangeable or transferable.

If we are trying to assign some other type of value to a variable we may get compile time error.

But in JavaScript we won't get any compile time error and type interchange is also possible.

Because JavaScript is a weakly types language and there is no strict type checking is performed and which is more forgiving language.

Note: In JavaScript a variable can contain any values in any point of the entire JavaScript Program.

Undefined:

This is a unique datatype which JS available only in JS and not available in any other language.

In order to understand the "undefined" datatype we should know the difference between "declaration" and "Definition" of variables.

Declaration:

UiBrains.com

Just declare a variable without any assignment is called declaration of a variable.

```
var name; // Declaration
```

Definition:

If assign some value to a declared variable we call it as variable definition.

```
name = "Naveen"; // Definition
```

“undefined” is a value which comes between declaration and definition.

```
var name; // undefined
name = "Naveen"; // Naveen
```

Before we assign a value to a variable, till then the value of that variable is “undefined”.

Once we assign a value to the variable then the value “undefined” will be replaced with the assigned value.

As in Java, once we declare a variable then some default variable will be assigned based on the type of the variable.

But in JavaScript there is no concept of type declaration at the time of declaring a variable. So for any kind of variable declaration a default value “undefined” will be assigned to it.

```
var firstName = 'Naveen';
console.log(firstName); // Naveen
```

Understanding Null:

As in JavaScript at the time of a variable declaration we have default value as “undefined”.

In order to assign a value to a variable after the declaration we can assign any value to that variable.

In JavaScript there is a provision to add some dummy value to the variable is called “null”.

This “null” is a value which is assigned to any type of variable in JavaScript.

Example:

```
var a;
a = null;
console.log(a); // null
```

Difference between Undefined & Null:

Undefined is value which is assigned at the time of declaration step and null is a value which is assigned at the time of definition step.

The default value before assignment of null or any value is "Undefined" and at the time of assignment there is an assignable dummy value which is null.

Example:

```
var name;
console.log(name); // undefined

name = null;
console.log(name); // null

name = 'naveen';
console.log(name); // naveen
```

Note:

For example let's take an application form where we need to fill in.

For any of the field we did not douched or filled the default value assigned to it is "undefined".

For any of the field which we add some dummy value as "Not Applicable" then the filed gets the default as null.

Summary of Variables & Types

At the time of declaration and definition of a variable we no need to declare the variable the type in JavaScript.

The same variable which can be assigned with any other types of values.

No Scoping information is available for variable declaration and definition.

JavaScript Basics

Here we will learn some of the basic concepts of JavaScript.

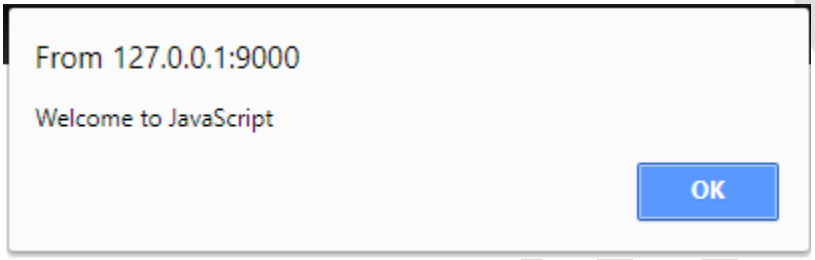
If we needs to execute any JavaScript code we don't required any HTML code base. We can just open the browser console and execute the following commands.

For logging information in JavaScript we have the following methods.

1) Alert Box
2) Confirm Box
3) Console log

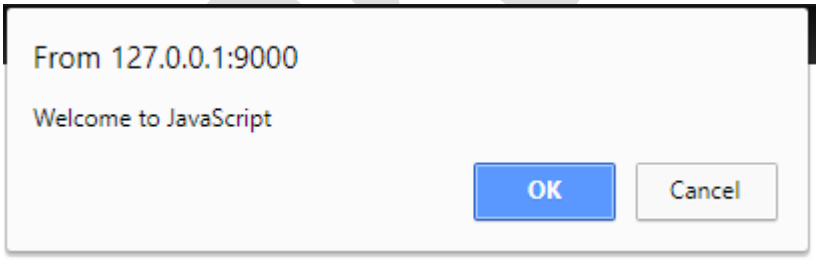
By Using alert we can show any textual information on the alert box.

```
alert("Welcome to JavaScript");
```



By Using Confirm box also we can show some information on the webpage

```
confirm("Welcome to JavaScript");
```

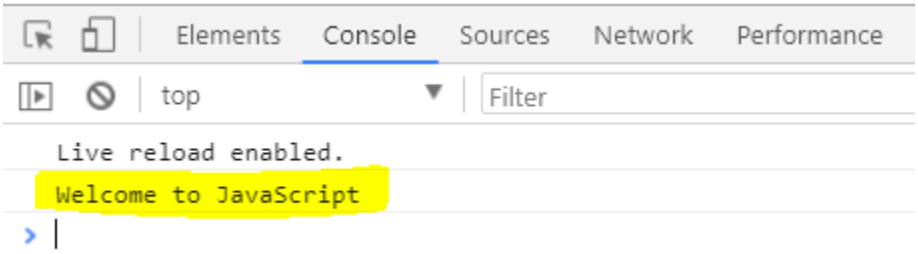


The Alert and Confirm boxes are displaying on the webpage. Each time we may have to click on ok button for each and every message on the box.

If we wants to add some logging related information, then alert and confirm boxes are not recommended.

For logging related information we may use console.log method of JavaScript. This prints the messages on the browser console instead of on the webpage as follows,

```
console.log("Welcome to JavaScript");
```



Console.log is the best way of printing any log related information, this prints the information on the browser console.

Along with the textual information we can display any kind of information on the console using Console.log method/function.

Let's discuss how to display the current date information on the webpage.

```
var date = new Date();
console.log("Today is " + date);
```

JavaScript Usage

We can use the JavaScript along with the HTML in the following ways,

1) Inline Way
2) Internal Way
3) External Way

Let's discuss about each of the way of using the JavaScript.

Inline Way

Here in this way we may mix the JavaScript code along with the HTML Code. However this approach is not recommended but we may use this approach for very few lines of code.

Example:

```
<div id="text-div">
    <p id="green-p">This is a Green Color Text</p>
    <button onclick="document.getElementById('green-p').style.color = 'blue'">Change to Blue</button>
    <button onclick="document.getElementById('green-p').style.color = 'red'">Change to Red</button>
</div>
```

Internal Way

In this way we may use separate <script> Tag and add the completed JavaScript code to it. This is one of the best way instead of mixing the JavaScript code with HTML codebase.

Example:

```html
<div id="text-div">
    <p id="green-p">This is a Green Color Text</p>
    <button onclick="blue();">Change to Blue</button>
    <button onclick="red();">Change to Red</button>
</div>

<script>
    function blue() {
        document.getElementById('green-p').style.color = 'blue';
    }
    function red() {
        document.getElementById('green-p').style.color = 'red';
    }
</script>
```

External Way

This is the best way of adding JavaScript to HTML codebase. Here we will be using an external ".js" file, this separates the HTML codebase with JS code base.

```html
// HTML File

<div id="text-div">
    <p id="green-p">This is a Green Color Text</p>
    <button onclick="blue();">Change to Blue</button>
    <button onclick="red();">Change to Red</button>
</div>

// Separate JS file

function blue() {
    document.getElementById('green-p').style.color = 'blue';
}
function red() {
    document.getElementById('green-p').style.color = 'red';
}
```

Variable Declaration in JavaScript

For any programming language the variables are heart of Programming. A variable which holds some data on the computer system and we may process that data using variables in the complete logic of any application.

Let's discuss about the variable declaration in JavaScript,

We can declare any variable in JavaScript using a keyword called "var"

In JavaScript we may not mention the datatype information of the variable like any oops programming languages.

Here we just use "var" keyword and assign any type of value to the variable. Based on the type of value we assigned, the JavaScript will automatically get the type information.

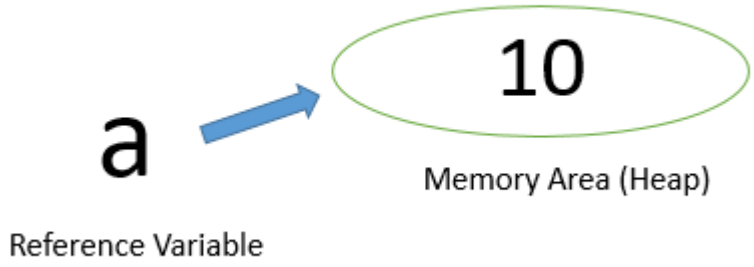
```
var a = 10;
var b = true;
var name = 'Naveen';

console.log(a);
console.log(b);
console.log(name);
```

Please note here we have not specified any datatype related information like string or number and Scoping related information like private, public.

Now deeply understand the variable declaration in JavaScript as follows,

When we declare a variable and assigned a value to it,

```
var a= 10;
```



When we declare a variable "a", "a" will be a reference variable which is pointing to a memory location (Heap Memory) which holds the value.

We may refer this value using the reference variable and use it for our mathematical calculations.

Data Types in JavaScript

In the variable declaration we can assign various types of values to the variables.

The type of value or the type of data which is assigned to a variable is called as datatype.

Let's discuss about the various datatypes available in JavaScript.

1) Number
2) String
3) Boolean
4) Undefined
5) Null

In JavaScript ECMA Script 5 version the above mentioned as the various datatypes are available.

Number Data Type

If we assign any numerical value or data to a variable is called as Number Datatype.

Normally in any other OOPS languages like Java, we may have various types of Numerical data like byte, short, int, long, float, and double based on the value of Numerical data.

But in JavaScript we do not have such kind of different types of types for Numerical data representation.

In JavaScript we can assign any Numerical data as a value to a variable, which can denoted with the datatype as Number.

Example

```
var a = 10;
var b = 20.5;
console.log(a); // 10
console.log(b); // 20.5
console.log(typeof a); // Number
console.log(typeof b); // Number
```

String Data Type

If we assign any textual value or data to a variable is called as String Datatype.

Normally in any language like Java, for any single character or letter they represent with a datatype called "Character" and any textual data which more than one letter or character they denote with the datatype called "String".

But in JavaScript there is no datatype called "Character", we can represent a single letter is also with the data type called "String".

Example:

```
1  var letter = 'a';
2  var name = 'Naveen';
3  console.log(letter); // a
4  console.log(name); // Naveen
5  console.log(typeof letter); // string
6  console.log(typeof name); // string
```

Boolean Data Type

If we assign any "true" or "false" value to a variable is called as Boolean Data Type.

Boolean data type contains only true or false value for any other language as well as in JavaScript.

Example:

```
1  var isEasy = true;
2  console.log(isEasy); // true
3  console.log(typeof isEasy); // boolean
```

Undefined Data type

This is one of the unique data type available only in JavaScript. If we just declare a variable and not assigned any value to it, then the default called "undefined" value will be assigned to the variable.

Example:

```
1  var a;
2  console.log(a); // undefined
3  console.log(typeof a); // undefined
```

Null Data Type

If we just declare a variable and not assigned any value to it, then the default value will be assigned to that variable is "undefined". If we declare a variable and assign a dummy value instead of actual value is "null" datatype.

Example:

```
1  var a = null;
2  console.log(a); // null
3  console.log(typeof a); // Object
```

In the Above Example, at Line Number 1, we have assigned a dummy value “null” instead of some meaningful value.

At the Line Number 3, we got the type of ‘a’ as Object instead of null. In JavaScript whenever we assign a value null to a variable then we get the type of “null” will be Object instead of null. This is one of the wired behavior of JavaScript.

So please remember type of ‘Null’ will be always Object but not “null” in JavaScript.

Operators in JavaScript

In any programming language for kind of mathematical operations we do require various types of operators.

Using operators we can perform some most of the mathematical calculations for writing any processing logic of any system.

Let’s discuss about the Operators in JavaScript.

Assignment Operator (=)

In JavaScript we represent the assignment operator using equal symbol. (=)

The Assignment is used to assign values to any variables.

Example

```
1  var name = 'Naveen';
2  console.log(name);
```

Arithmetic Operator (+ , - , / , * , %)

The Arithmetic operators are used to perform some mathematical operations like addition, subtraction, division, multiplication and modulus/remainder.

Example

UIBrains.com

```
var a = 10;
var b = 20;

var sum = a + b;
console.log(sum); // 30

var sub = a - b;
console.log(sub); // -10

var mul = a * b;
console.log(mul); // 200

var div = a / b;
console.log(div); // 0.5

var mod = a % b;
console.log(mod); // 10
```

Short hand Math (+= , -= , *= , /= , %=)

The short hand math operators are used to apply shortcut mathematical operations.

If we wanted to add two values and assign to a variable we may use this short hand math operators.

```
1 var a = 10;
2 var b = 20;
3 var sum = 0;
4 sum = sum + (a + b);
5 console.log(sum); // 30
```

If our math operation is like the above mentioned line number 4, then for such cases we may this short hand math as follows,

```
1 var a = 10;
2 var b = 20;
3 var sum = 0;
4 sum += (a + b);
5 console.log(sum); // 30
```

Conditional Operators (< , > , <= , >= , !=)

The conditional operators are used to check logical conditions on various values of variables.

We normally use these conditional operators to evaluate some logical expressions.

```
var age = 30;
if(age <= 30){
    console.log('Get a Job');
}
else{
    console.log('Get Marry Soon');
}
```

Here we are evaluating the logical expression , if the age is less than or equal to 30 , then "Get a Job" will be printed on the console, if not then "Get Marry Soon" will be printed on the console.

Unary Operator (++ , --)

All the above operators required more than one operand or variable to apply the operator, but for this it require only one variable is called unary operator.

The ++ , -- operator is also called as increment and decrement operators.

If our requirement is for incrementing or decrementing any variable value by '1' we use this operator.

```
1 var a = 10;
2 a = a + 10; // same
3 a += 10; // same
4 a++; // same
```

In the above example, at line number 2 , 3 , 4 all are same, means by using all these ways we can increment the 'a' value by 1.

For this kind of requirement the unary operator is the shortest way.

Logical Operator (&&, ||)

The logical operators are used to check some logical conditions.

&& specifies if both the values are true then only the result will be true;

|| specifies if aby one value is true then the result will be true;

Example

```
var courseCompleted = true;
var practiceCompleted = true;
if(courseCompleted && practiceCompleted){
    console.log('You will definitely get a Job');
}
else {
    console.log('Keep trying');
}
```

String Concatenation Operator (+)

The String Concatenation Operator is used to append two strings or a string and a number.

We normally use '+' symbol to add two numbers, the same operator is also acts like a String concatenation operator while we are trying to add a String to a number or both the strings.

Example:

```
var a = 10;
var b = '20';
var append = a + b;
console.log(append); // 1020

var str1 = 'Good';
var str2 = 'Morning';
var greeting = str1 + str2;
console.log(greeting); // Good Morning
```

Ternary Operator (? :)

This operator is used as an alternative to if-else condition.

The syntax of ternary operator is as follows,

```
(condition) ? trueSection : falseSection;
```

First we evaluate the condition, if it is true then the true Section will be executed and if the condition is false then the false section will be executed.

```
var age = 18;
var category ;
(age <= 18) ? category = 'Minor' : category = 'Major';
console.log(category); // Minor
```

Typeof Operator

This operator is used to check the type of a value is assigned to any variable.

Normally we use this operator to check the datatype of a value assigned to a variable.

Example

```
var a ;
console.log(typeof a); // undefined

a = 10;
console.log(typeof  a); // number

a = 'Naveen';
console.log(typeof a); // string

a = true;
console.log(typeof a); // boolean

a = null;
console.log(typeof a); // object
```

Note: If a variable is just declared and not assigned any value then the default value assigned to that variable is 'undefined'.

== Operator

In JavaScript the == operator is used to check the equality of two operands.

If we use this operator to check the equality of two operands first it will do the automatic type conversion and then apply the type conversion.

Example:

```
var a = 10;
var b = '10';
if(a == b){
    console.log('Both are Equal');
}
else {
    console.log('Both are NOT Equal');
}

// Output : Both are Equal
```

In the above example we have a number and String. The == operator first check for the type conversion possible or not and do the checks for the equality. Here it prints on the console as "Both are equal".

Even though the content is same but both are from different datatype. So normally if both are of different datatype forget about the content, they are always not equal only.

Note: we should normally check the equality of two same data typed values only.

Due to this wired behavior of automatic type conversion and checks for the equality , this approach of using == operator for use of equality is not elegant way.

We can resolve this by using === operator.

=== operator

This is the best way of checking the equality two operands. Here there is NO automatic type conversion will be happened like == operator.

In JavaScript always use only this operator to check the equality of two operands.

Example:

```
var a = 10;
var b = '10';
if(a === b){
    console.log('Both are Equal');
}
else {
    console.log('Both are NOT Equal');
}

// Output : Both are NOT Equal
```

In the above example, it will print on the console as "Both are NOT equal".

Conditional Statements

In JavaScript we use various conditional statements to check some mathematical conditions to providing processing logic for any system.

The conditional statements / Loop statements are as follows,

1) If Condition
2) If else Condition
3) For loop
4) While loop
5) Do while loop
6) Switch statement

Let's discuss about these conditional statements,

If Condition

The 'if' condition is used to evaluate the logical condition if any condition is matched or results to true.

Example:

```
1  var isJSEasy = true;
2  if(isJSEasy){
3      console.log('JavaScript is Easy');
4  }
5
6  //Output : JavaScript is Easy
```

For have to specify the logical condition, if the condition is true then the block of code for 'if' statement will be executed.

This is just one way of checking the condition. If the condition is not met or if the condition is false then nothing will be done. So for this kind of cases we may use 'if-else' statements.

If-Else Condition

The 'if-else' is one of the best way checking for true part and false part also for checking the conditions for any processing logic.

If the condition is true then if condition will be executed and If the condition is false then the else statement will be executed.

Example

```
var a = 10;
var b = 20;
if(a > b){
    console.log('a is big');
}
else {
    console.log('b is big');
}

// Output : b is big
```

For Loop

The for loop is used for looping through some conditions, until the condition is true the loop will be continued, once the condition is false then the control comes out from the for loop.

The Syntax of for loop is,

```
for(initialization ; condition ; increment / decrement){
    // statements
}
```

Each for loop contains 3 sections as specified as above like initialization, condition and increment / decrement section.

Initialization section will be executed only once, and remaining sections will be executed until the condition is true.

Let's take an example of printing 1 – 10 values on the console using for loop.

```
for(var i = 1 ; i <=10; i++){
    console.log(i);
}

// Output :  1 2 3 4 5 6 7 8 9 10
```

While Loop

While loop is also a looping statement which loop through the condition and until the condition is true the block of while loop will be executed and once the condition is false then only the control comes out from while loop.

Syntax:

```
// variable declaration
while(condition){
 // statements , increments / decrements
}
```

Let's see an example of printing 1 – 10 values using while loop.

Example:

```
var i = 1;
while(i <= 10){
    console.log(i);
    i++;
}

// Output :  1 2 3 4 5 6 7 8 9 10
```

Note: We should be very careful while using while loop, if we didn't add the increment or decrement statement the loop may fall into infinite loop.

Do While Loop

We will use do while loop is also for looping purpose. The difference between while and do while is this 'do while' loop will be executed at least once.

Syntax:

```
// variable creation
do{
    // statements or increment or decrement
}
while(condition);
```

Let's see an example of do while loop to print 1 – 10 values.

Example:

```
var i = 1;
do{
    console.log(i);
    i++;
}
while(i <= 10);

// Output :  1 2 3 4 5 6 7 8 9 10
```

Note: While using do while loop, we should always add increment or decrement of variable otherwise do while loop may fall into infinite loop.

Switch Statement

This is an alternative to if-else ladder statement. Instead of using many if-else conditions we may use this switch statement.

Syntax:

```
// variable declaration
switch(condition){
    case value:
        // statements
        break;
        .............
     default:
     // statements
     break;
}
```

In the switch statement once the condition is match then the respective case of statements will be executed. If none of the case matches then finally the 'default' case will be executed.

We should always use 'break;' statement for each of the 'case' otherwise all the cases will be executed until it reaches the break statements.

Example:

```
var day = new Date().getDay();
var actualDay = '';
switch(day){
    case 0:
        actualDay = 'Sunday';
        break;
    case 1:
        actualDay = 'Monday';
        break;
    case 2:
        actualDay = 'Tuesday';
        break;
    //...........
    default :
        actualDay = 'Who Knows ?';
        break;
}
console.log(actualDay);

// Output :  Monday
```

Objects in JavaScript

JavaScript is an object oriented programming language. Here in JavaScript we will can provide solutions for any requirements in Objects oriented way instead of a functional programming.

The object orientation of JavaScript is different than any other Object Oriented programming languages. In any other languages we will a have a class based structure or blue print and creates Objects for that class and the complete data of the system is evolved in objects.

But here in JavaScript we do not have a concept called classes as of ES5 version.

We can create objects directly without classes.

Let's understand the ways of creating objects and adding properties to objects and deleting properties of objects and deleting properties of an Objects.

We will create objects in JavaScript in simple way as follows,

```
var obj = {};
console.log(obj);

// Output : Object { }
```

We can add properties to an objects even after creation objects at runtime as follows,

```
    var employee = {};
    employee.firstName = "Naveen";
    employee.lastName = "Saggam";
    employee.designation = "Team Lead";
    console.log(employee);

```

The Output will be printed on the console as follows

```
▸ Object { firstName: "Naveen", lastName: "Saggam", designation: "Team Lead" }
```

Here in the above example we have created an empty objects and added all the properties to it at runtime. We can also add the properties to an Object at the time of creation also. This is called as Object Literal.

Let's see the Objects Literal Example as below,

```
    var employee = {
        firstName : 'Naveen',
        lastName : 'Saggam',
        Designation : 'TeamLead'
    };
    console.log(employee);
```

Now let's understand the ways of retrieving properties of an object.

We can access the properties of an Objects using two ways as follows,

1) Using Dot Notation
2) Using Brackets Notation.

Using Dot Notation

We normally access the properties of an object using dot notation. The syntax of Dot notation is as follows,

```
1     ObjectName.propertyName;
```

Example:

```
1 console.log(employee.firstName); // Naveen
2 console.log(employee.lastName); // Saggam
3 console.log(employee.designation); // Team Lead
```

Brackets Notation

We can access the Object's properties using brackets notation also.

The syntax of using brackets notation is as follows,

Syntax:

```
1 ObjectName["propertyName"];
```

Let's access the properties of employee object using brackets notation.

Example:

```
1 console.log(employee["firstName"]); // Naveen
2 console.log(employee["lastName"]); // Saggam
3 console.log(employee["designation"]); // Team Lead
```

Note:

We normally use this brackets notation to retrieve invalid property names an objects like any property name starts with number. But invalid property names are not allowed inside an objects.

We use this brackets notation to access the properties of an Array. This we will discuss in Arrays concept.

We also use this brackets notation to access dynamic values of an objects , for accessing dynamic values we can't use the dot notation.

Let's see the example of access dynamic values of an Object.

```
var student = {
    name : 'John',
    age : 20,
    course : "Computers"
};
var studentName = "name";
console.log(student.studentName); // undefined
console.log(student[studentName]); // John
```

Note: In order to access the properties of an Object using Dot notation we must use the same name as the property name which is exists in the object. We cannot access the dynamic values using Dot notation. We can access the dynamic values of an objects using Brackets notation.

We have discussed about accessing the existing properties of an object. But what happens we are trying to access the properties which are not existing in an objects.

Normally for any other OOPS languages we may get some error. But here in JavaScript we won't get any compile time error but it simply returns "undefined".

Let's see the example of accessing non existing properties of an Object as follows,

```
var student = {
    name : 'John',
    age : 20,
    course : "Computers"
};
console.log(student.address); // undefined
```

Note: In the above Student object there is no property called "address", so we trying to access the non-existing property of an object in JavaScript we will get "undefined".

We can even create objects inside other objects is called as nested Objects.

Let's understand the Nested Objects and the way of access the properties of nested objects.

Nested Objects

A JavaScript object which is defined in side another JavaScript object is called as Nested Object. This is a very common scenario we may in almost all OOPS programming languages.

Example:

```
var student = {
 name : 'Rajan',
 age : 22 ,
 dept : 'engg',
 address : {
     city : 'Hyderabad',
     state : 'Telangana',
     country : 'India'
 }
};

```

In the above Example we have a student Object and inside we have another object called 'address'.

We can access the nested objects properties also just like any other property of an object.

Let's access the city property of nested object 'address' which is inside of student object.

```
var city = student.address.city;
console.log(city); // Hyderabad
```

We can even add some few extra properties to the nested object 'address' like the normal way of adding properties to an object.

```
student.address.street = "Hitech City";
```

We can do the console log the entire student object to see whether the 'street' property is added to 'address' object or not as follows,

```
console.log(student);
```

The output is as follows,

```
{…}
  address: Object { street: "Hitech City", city: "Hyderabad", state: "Telangana", … }
  age: 22
  dept: "engg"
  name: "Rajan"
  __proto__: Object { … }
```

Arrays in JavaScript

An Array is a very common concept available in almost all the programming languages.

An Array in JavaScript is an indexed collection of values of various datatypes.

Array is a 'zero' index based in all the programming languages.

Let's discuss the declaration of array and adding properties to an array and accessing the properties / elements of an array and also the default functions available to various operations with arrays.

Creation of arrays:

```
// creation of array
    var myArray = [10,20,30,40];

// Accessing an array and its properties
    console.log(myArray[0]); // 10
    console.log(myArray[3]); // 40
```

We can add properties to an array using based on the index of the array only.

```
myArray[4] = 50; // added new property
console.log(myArray);
```

We can access the length of the array using 'length' property of an array.

```
console.log(myArray.length); // 5
```

Let's discuss some of the default available function to perform various operations on elements of an array.

reverse ()

This method / function is used to reverse the elements of an array.

The usage of reverse function is as follows,

```
myArray = [10,20,30,40,50];
console.log(myArray); // Array [ 10, 20, 30, 40, 50 ]
myArray.reverse(); // to reverse the array
console.log(myArray); // Array [ 50, 40, 30, 20, 10 ]
```

Shift()

This method / function is used to remove the first element of an array.

The usage of shift function is as follows,

```
myArray = [10,20,30,40,50];
console.log(myArray); // Array [ 10, 20, 30, 40, 50 ]
myArray.shift();
console.log(myArray); // Array [ 20, 30, 40, 50 ]
```

Unshift()

This method / function is used to add a specific element (or elements) to the front of an array.

The usage of unshift() is as follows,

```
myArray = [10,20,30,40,50];
console.log(myArray); // Array [ 10, 20, 30, 40, 50 ]
myArray.unshift(80,90);
console.log(myArray); //Array [ 80, 90, 10, 20, 30, 40, 50 ]
```

Pop()

This method / function is used to remove the last element of an array.

The usage of pop() is as follows,

```
myArray = [10,20,30,40,50];
console.log(myArray); // Array [ 10, 20, 30, 40, 50 ]
myArray.pop();
console.log(myArray); // Array [ 10, 20, 30, 40 ]
```

Push()

This method / function is used to add an element (or elements) to the end of an array.

The usage of push() is as follows,

```
myArray = [10,20,30,40,50];
console.log(myArray); // Array [ 10, 20, 30, 40, 50 ]
myArray.push(100,200);
console.log(myArray); // Array [ 10, 20, 30, 40, 50, 100, 200 ]
```

Splice()

This method / function is used to remove an element from an array from the specified index position.

Syntax : splice(index,n);

Here 'index' means the index of the removable element and 'n' specifies the number of elements should be deleted from that index position.

The usage of splice() is as follows

```
myArray = [10,20,30,40,50];
console.log(myArray); //Array [ 10, 20, 30, 40, 50 ]
myArray.splice(1,3); // removes 20 , 30 , 40
console.log(myArray); // Array [ 10, 50 ]
```

Let's see the example which removes a single element from an array.

```
myArray = [10,20,30,40,50];
console.log(myArray); // Array [ 10, 20, 30, 40, 50 ]
myArray.splice(1,1); // removes 20
console.log(myArray); // Array [ 10, 30, 40, 50 ]
```

Slice()

This method / function is used to create a new copy of an existing array and typically assigned to a new variable.

The usage of slice() is as follows,

```
myArray = [10,20,30,40,50];
console.log(myArray); // Array [ 10, 20, 30, 40, 50 ]
var myArray2 = myArray.slice();
console.log(myArray); // Array [ 10, 20, 30, 40, 50 ]
```

Indexof ()

This method / function returns the index of any element. Here we can even specify the Index position also in this method, this returns the first element that matches with that Specified index position.

Syntax: myArray.indexof (search, index);

The usage of Indexof () is as follows,

```
myArray = [10,20,30,40,50];
var result = myArray.indexOf(30, 0);
console.log(result); // 2
```

Join()

This method / function is used to return a string representation of all the array elements joined with the specified parameter.

The usage of join() is as follows,

```
myArray = [10,20,30,40,50];
var output1 = myArray.join(" * ");
console.log(output1); // 10 * 20 * 30 * 40 * 50

myArray = [10,20,30,40,50];
var output2 = myArray.join(" - ");
console.log(output2); // 10 - 20 - 30 - 40 - 50
```

Note: we do have lot of methods / functions available for array, Please have a look at the below URL for complete information about an Array of JavaScript.

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array

Functions in JavaScript

For any programming language functions / Methods place a major role in logic development.

A Function is a block of code which takes in various arguments and internally does some processing and finally returns the result.

A function in any programming language is treated as an engine, which takes in some parameters and does some processing and finally returns the result.

One function calls other function and gets the results from other function does some more process based on the result got from the previous function. This is called as function chain.

In order to build any software system we do require functions for processing logic and objects for holding the data of system, functions uses these objects data and proceeds for more processing of any system.

Let's understand the functions in JavaScript.

We can create a simple function in JavaScript with no arguments as follows,

```
function greet() { // function definition
    var output = "Good Evening";
    console.log(output); // Good Evening
}

greet(); // function Execution
```

Note: For any function there are always two parts, one is function definition and second is the function execution.

Just by declaring the function definition which cannot be executed automatically, we needs to call / execute the function.

Now let's understand a creation of a simple function with arguments,

```
1  // function definition
2  function greetArgs(name) {
3      var output = "Good Evening " + name;
4      console.log(output); // Good Evening Naveen
5  }
6
7  // function Execution
8  greetArgs("Naveen");
```

Note: Here at line Number 8, we are passing "Naveen" parameter to the greetArgs() function and which takes in the name and prints it on the console.

Executing a function with less number of parameters

In any other programming languages other than JavaScript we must have to call / execute the function with the same number of parameters as mentioned in the function definition, otherwise we may get compile time error.

But in JavaScript we call a function with less number of arguments than the number of parameters specified in the function definition. In this case we won't get any error but simply the remaining parameter gets the value undefined.

Let's discuss the example as below,

```
function lessArgs(name,age) {
    var output = " Your name is : " + name + " age : " +age;
    console.log(output); //  Your name is : Naveen age : undefined
}

lessArgs("Naveen");
```

In the above example the function accepts two parameters like name, age. But while calling / executing the function we provided only one parameter 'name' then we wouldn't get any error and the age parameter get the value '*undefined*'.

Executing a function with more number of parameters

We can even execute a function by providing more number of parameters than the expected number of parameters of the function.

If we pass more number of parameters to a function than the expected number of parameters, then we won't any error in JavaScript and simply the extra parameters will be ignored by the function.

Let's discuss the example as below,

```
function moreArgs(name,age) {
    var output = " Your name is : " + name + " age : " +age;
    console.log(output); // Your name is : JOHN age : 50
}

moreArgs("JOHN",50,50000);
```

In the above example the moreArgs () function is expecting only two parameters like name, age. But while calling the function we are providing name, age, salary also.

In this case the salary parameter will be ignored by the function, and we won't get error in this case.

Note: Based on the previous scenarios a single function in JavaScript can accepts more or less number of parameters also. This implies that function overloading is not possible in JavaScript.

In any other OOPS languages a function can accepts the exact parameters as specified in the function definition. We can't pass more or less number of parameters. For this reason we may write the overloaded functions to accept the more or less number of parameters for the same function name.

But this is not required in JavaScript. So it clearly states that function overloading is not possible in JavaScript.

In JavaScript function names are unique, we should not write more number of functions with the same name in same JavaScript file. Multiple function with same name is not allowed in JavaScript.

Functions with Return Values

As of now we have seen the function without return values. Now will understand the JavaScript functions with return values. Normally for any function in JavaScript we may not specify the return value type in the function declaration. We just return any type of value as a result.

Let's discuss the example as follows,

```
1 function greetReturn(name) {
2     var output = "Good Evening " + name;
3     return output;
4 }
5 var myOutput = greetReturn("Naveen");
6 console.log(myOutput); // Good Evening Naveen
```

In the above example the greetRetun () function is returning 'output' as a result return value. We are capturing that value at line number 5 and printing it on the console.

If any function is returning with some value, then we must have to capture that return value and use it for printing to console or use it for further processing.

Function with Empty return

If any function is return with an empty value then result of it will be undefined.

Let's discuss a function with empty return value,

```
function emptyReturn(name) {
    var output = "Good Evening " + name;
    return; // empty return
}

myOutput = emptyReturn("Naveen");
console.log(myOutput); // undefined
```

As specified in the above function if we are trying to capture the empty return, then the captured value contains the value *undefined*.

Function Expression

This is the feature of functional programming. Here we can assign the function itself as a value to a variable.

If we assign a function itself as a value to a variable is called as function Expression or First Class Function. JavaScript is contains full of first class functions.

Let's discuss about the First Class Function / Function Expression as follows,

```
// function Expression
var greetMe = function greet() {
        var output = "Good Evening";
        console.log(output); // Good Evening
};

//function Execution
greetMe();
```

In the above example we have created a variable "*greetMe*' and assigned a function itself as a value to it. We can execute that function expression with the variable name only but not with the function name.

Anonymous Function Expression

Any function expression without a name is called as Anonymous function Expression.

In the previous example we have declared a function Expression and we have executed it with the variable name only. So here we never use the function name then function name is dummy.

So we can create a function expression without a function name as follows is called as Anonymous function expression.

```
// Anonymous Function Expression
var greetMe = function() {
    var output = "Good Morning Naveen";
    console.log(output); // Good Morning Naveen
};

// Function Execution
greetMe();
```

Anonymous Function Expression with Arguments

In the above example we discussed about anonymous function without arguments. Now let's discuss the anonymous function with arguments.

```
// Anonymous Function Expression with args
var anomymousFnArgs = function (name) {
    var output = "Good Evening " + name;
    console.log(output);
};

// Function Execution
anomymousFnArgs("Naveen");
```

Functions as Arguments without Arguments

In order to build a software system or product we should have to use various functions and share data between those functions.

Each function accepts various arguments and process the data and returns a results back to the caller function.

As of now we discussed a function accepts simple arguments like string or numbers, now let's discuss the functions with accepts functions as arguments.

```
1  function greetMe1() {
2      var output = "Good Evening 1";
3      console.log(output);
4  }
5
6  function greetMe2() {
7      var output = "Good Evening 2";
8      console.log(output);
9  }
10
11 function greeter(fnName) { // function as args
12     fnName(); // calling the function
13 }
14
15 greeter(greetMe2); //Good Evening 2
16 greeter(greetMe1); //Good Evening 1
```

In the above example the *greeter*() function takes in the parameter as a function name itself and it executes the passed in function.

If we pass '*greetMe2*' to '*greeter*()' function at line number 15 , so the line number 12 will execute the '*greetMe2*()' function.

This is a very common scenario in JavaScript programming.

Functions as Arguments with Arguments

As of now we discussed about the functions accepts an arguments as functions itself but those are without arguments. Now let's understand the functions are arguments with arguments.

```
function greetHim1(name) {
    var output = "Good Evening " + name;
    console.log(output);
}

function greetHim2(name) {
    var output = "Good Evening " + name;
    console.log(output);
}

function greeterEngine(fnName,name) {
    fnName(name);
}

greeterEngine(greetHim1,"Naveen"); // Good Evening Naveen
greeterEngine(greetHim2,"John"); // Good Evening John
```

Functions inside an Objects

In the previous sections we have discussed about Objects creation, accessing the properties of an objects now let's discuss the Objects which contains functions inside of it.

This is also a very common scenario where we have an objects which contains various functions inside of them.

```
// Object contains a function
var employee = {
    firstName : "Naveen",
    lastName : "Saggam",
    fullName : function () {
        var output = this.firstName + " " + this.lastName;
        console.log(output);
    }
};

// Execution of a function inside an Object
employee.fullName(); // Naveen Saggam
```

Math Object

Math Object is useful for performing various mathematical operations in developing any software system. Math Objects contains various properties to use it for any mathematical operations.

Let's understand the usage of Math Objects below,

Math.PI

Math.PI property is used to get the PI value. The usage is as follows,

```
var PI_VALUE = Math.PI;
var output = "The PI Value is : " + PI_VALUE;
console.log(output); //The PI Value is : 3.141592653589793
```

Math.min()

This function of Math Object is used to find the min value of the specified number of parameters.

Example:

```
var minValue = Math.min(10,20,30,40);
output = " Min of 10,20,30,40 is : " + minValue;
console.log(output); //Min of 10,20,30,40 is : 10
```

Math.max()

This function of Math Object is used to find the max value of specified number of parameters.

Example:

```
var maxValue = Math.max(10,20,30,40);
output = " Max of 10,20,30,40 is : " + maxValue;
console.log(output); //Max of 10,20,30,40 is : 40
```

Math.pow()

This function of Math Object is used to find 'x' to the power of 'y' results.

Example:

```
var pow2_4 = Math.pow(2,4);
output = " 2 ^ 4 is : " + pow2_4;
console.log(output); // 2 ^ 4 is : 16
```

Math.random()

This function of Math Object is used to get a random value between 0 and 1. This method we can use to generate any random values of any range.

Example: To generate random numbers from 0 to 1000 as follows,

```
var random = Math.round(Math.random() * 1000);
output = " The Random Value : " + random;
console.log(output); // The Random Value : 834
```

Note: to know more about Math Object please have a look at the below link.

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Math

Date Object

The Date object is used to work with Date and Time related calculations.

This Date Object's properties and functions are very much useful to get the time and dates of various time Zones.

Let's understand the properties and functions of Date Object and their usage.

In order to use Date Object, first we have to create a Date Object.

Example:

```
var date = new Date();
```

Get Today's Date

```
var today = new Date();
output = " Today is : " + today;
console.log(output);
// Today is : Wed Mar 28 2018 10:59:19 GMT+0530 (India Standard Time)
```

Get Date of the Month

```
var today = new Date();
var date = today.getDate();
output =" The Date is : " + date;
console.log(output); // The Date is : 28
```

Get Day of a Week

```
var today = new Date();
var dayOfWeek = today.getDay();
output =" The Day is : " + dayOfWeek;
console.log(output); // The Day is : 3
```

Get Full Day of the week

```
var today = new Date();
var dayOfWeek = today.getDay();
switch(dayOfWeek){
    case 0:
        fullDay = "Sunday";
        break;
    case 1:
        fullDay = "Monday";
        break;
    case 2:
        fullDay = "Tuesday";
        break;
    case 3:
        fullDay = "Wednesday";
        break;
    default:
        fullDay = " Nothing ";
        break;
}
output = "The Full Day is : "+ fullDay;
console.log(output); // The Full Day is : Wednesday
```

Note: To Know more about Date Object please have a look at the below link

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Date

Dom Manipulation

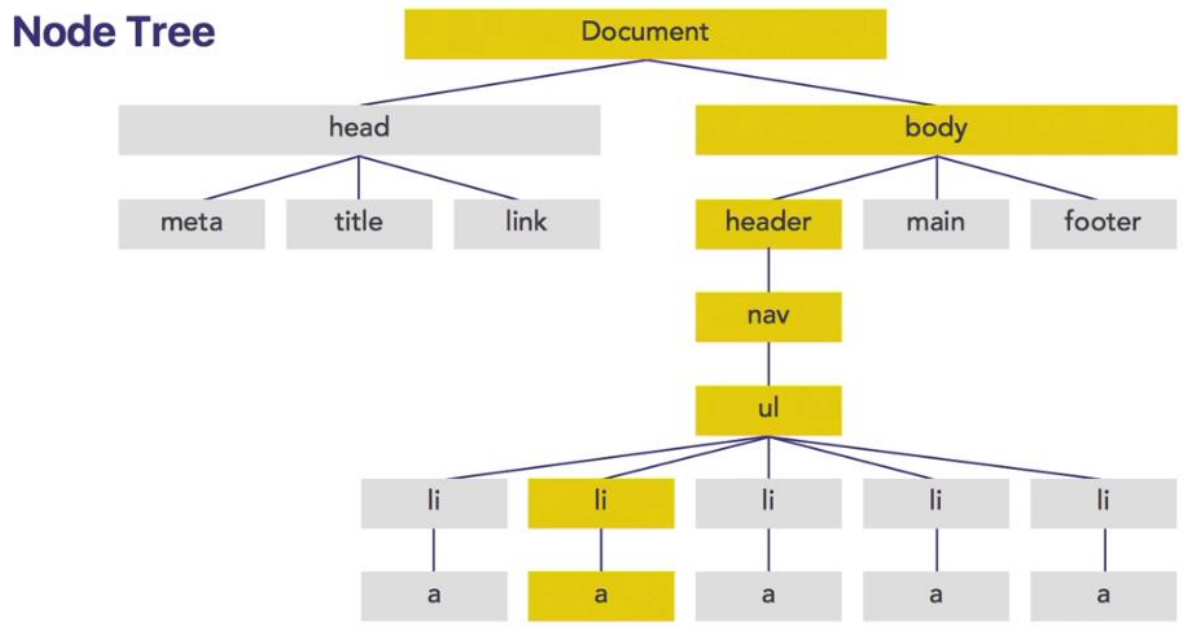
The JavaScript is used for browser side scripting and to provide dynamic nature to any website.

JavaScript provides dynamic nature to any webpage using the concept called DOM Manipulation.

First let's understand the concept of DOM (Document Object Model).

Whenever browser gets a HTML file before displaying the content on the browser, first it reads the entire HTML page and prepares a tree like structure using all the HTML tags / elements. This Tree like structure is called DOM Tree. Each html element inside the DOM Tree is called as 'DOM Node'.

Example:



The above diagram denotes a Node Tree or DOM Tree.

Once this tree structure is built, then the browser renders each html element / tag and display them on the browser / webpage.

How the JavaScript provides the dynamic nature to HTML is, by using JavaScript we will fetch each of the HTML tag / element / DOM Node from the DOM Tree and adds our own content so that browser parses the new DOM Tree nodes and displays them on the browser.

This DOM Tree of Nodes are responsible for the display of content on the webpage. So in the backend by using JavaScript we will change the DOM Nodes using the Concept called DOM Manipulation then we will get the Dynamic view on the webpage.

By Using JavaScript we can do the following to the DOM Tree.

1) Change the existing DOM Node Content
2) Add a new DOM Node to the DOM Tree
3) Remove a DOM Node from the DOM Tree

Note: Please note HTML Element, HTML Tag, DOM Node all denotes the same.

So for each HTML Document there will be one DOM Tree. So by using JavaScript we can change the HTML Content to be displayed on the webpage.

By using JavaScript we can change not only DOM Nodes but also Browser related properties.

In a Browser each tab is represented with one DOM Tree. A Browser itself is also an Object and it contains a collection of DOM Trees.

This is called as Browser Object Modal.

In Browser Object Model the browser itself is treated as an Object and it contains a long list of Objects as follows,

1) Document (DOM Tree)
2) Navigation buttons
3) URL

All these Objects are modeled together and is called as BOM (Browser Object Model).

NOTE: In the BOM Model the top level object is 'window' and it represents the entire Browser itself.

In the DOM Model the top level object is 'Document' and it represents the entire Document itself.

Let's discuss some of the operations that we can perform on BOM Objects by using JavaScript.

```
window.innerWidth; // to access the width of the Browser

window.innerHeight; // to access the height of the Browser

window.open(); // to open a new Tab

window.document; // to access the Document object
```

We can access the Document Object using either 'window. document' or directly by using 'document'.

Let's understand the DOM Manipulation using JavaScript as follows,

We can manipulate the DOM Nodes using the Document Object's properties and methods/functions.

DOM Properties

The following are the various DOM properties to be accessed by JavaScript.

```
// To access the complete body element
var body = document.body;

// To access the title of the current webpage
var title = document.title;

// to access the URL of the current webpage
var url = document.URL;
```

DOM Methods / Functions

The following are the various methods to be used by JavaScript to manipulate the DOM nodes.

```
// to select a HTML Element with the specific ID (main-nav)
document.getElementById('main-nav');

// to select all the HTML Elements with the class name (container)
document.getElementsByClassName('container');

// to select all the HTML Elements using the HTML tag (img)
document.getElementsByTagName('img');

// to select the first element matching the selector (.main-nav a)
document.querySelector('.main-nav a');

// to select all the HTML elements matching the selector (.container p)
document.querySelectorAll('.container p');
```

Let's discuss some of the examples

```
<div id="text-div">
    <p id="green-p">This is a Green Color Text</p>
    <button onclick="blue();">Change to Blue</button>
    <button onclick="red();">Change to Red</button>
</div>

<script>
    function blue() {
        document.getElementById('green-p').style.color = 'blue';
    }
    function red() {
        document.getElementById('green-p').style.color = 'red';
    }
</script>
```

In the above example we will change the text color dynamically while pressing blue , red buttons using DOM Manipulation.

Example:

```
<div id="image-div">
    <img src="../img/google.jpg" width="300px" height="200px" id="myImage">
    <br>
    <button onclick="facebook();">Facebook</button>
    <button onclick="youtube();">Youtube</button>
</div>
```

```
function facebook() {
    document.getElementById('myImage').setAttribute('src','../img/facebook.jpg');
}

function youtube() {
    document.getElementById('myImage').setAttribute('src','../img/youtube.jpg');
}
</script>
```

In the Above Example if click on 'Facebook' button we will get the Facebook image and if we click on YouTube button we will get the YouTube image dynamically by using JavaScript DOM manipulation.

JavaScript DOM Events

Any kind of operation or an action that can be performed on a browser is called as an event.

The moment a user is performing an action on a browser is an event.

Let's discuss some of the DOM Events,

1) Opening a browser is an event
2) Open address bar and enter URL is an event
3) User clicks on a link is an event
4) Move a mouse cursor is an event
5) Click on a button is an event
6) Focus on an input field is an event.

Mouse Events	Keyboard Events	Form Events	Document Events
Click	Keypress	Submit	Load
Dbclick	Keydown	Change	Resize
Mouseenter	Keyup	Focus	Scroll
mouseleave			unload

JS DOM Event Handling

In JavaScript we can perform the DOM Manipulation using various event handling methods as specified above.

The Actual Event handling event handling works as follows,

1) Find a node to which we trigger an event
2) Find the events of the node
3) Write a script to trigger an event for that node.

Example:

```
 HTML Code
-----------------
<div id="image-div">
    <img src="../img/google.jpg" width="300px" height="200px" id="myImage">
    <br>
    <button onclick="facebook();">Facebook</button>
    <button onclick="youtube();">Youtube</button>
</div>

JavaScript Code
------------------
function facebook() {
    document.getElementById('myImage').setAttribute('src','../img/facebook.jpg');
}

function youtube() {
    document.getElementById('myImage').setAttribute('src','../img/youtube.jpg');
}
```

JS DOM Event Listeners

In the process of DOM Manipulation first we will get a html element and hook up and event method and we can perform some operations on it.

In the process of event listeners, we normally hook up an event listener to the specific HTML element and attach an event handling method to it. If the specified event happens then the hooked up event handling method will be executed.

Syntax of Event listener is as follows,

```
htmlElement.addEventListener('event_name',event_handling_function);
```

Let's discuss an example of Event Listeners,

```
1   HTML Code
2   ------------
3
4  <textarea id="text-area" rows= '5' cols='10'></textarea>
5
6  JavaScript Code
7  ------------------
8  var textArea = document.querySelector("#text-area");
9
10 textArea.addEventListener('keypress',start);
11
12 function start() {
13 //statements of event handling
14 }
```

In the above example of Event Listeners way of event handling works as follows,

1) First we needs to fetch the required HTML Element using any DOM manipulation methods.(line no : 8)
2) We needs to hookup an event handling method to it along with the event name and event handling function name. (line no: 10)
3) In the last we have define a method / function for handling that event.

Note: Whenever the 'keypress' event happened for the text-area then start() will be executed.

JavaScript Scopes

JavaScript Scopes & Closures is one of the important concept to learn in JavaScript. Without these concepts also we can write the JavaScript code, but once we learn these concept code we may write more elegant code and most of the bug free code.

Now let's understand the concept of scopes and then closures concept.

Scopes & Block Scoping

Scopes concept is not only specific to JavaScript, these kind of terms we can even see in other OOPS languages.

A scope is considered as a block where the accessibility of variables exists. The variables declared inside the block are not accessible outside.

Example:

```
1  var a = 10;
```

If we declare a variable somewhere in the middle of a program, can we guess what are all the places it is accessible?. It is purely depend on the scope where we declare that variable is declared.

If the variable declared in the global scope, It will be accessed throughout the program. If the variable is declared in any block, then we can access the variable inside of that block only, outside of that block it is not accessible.

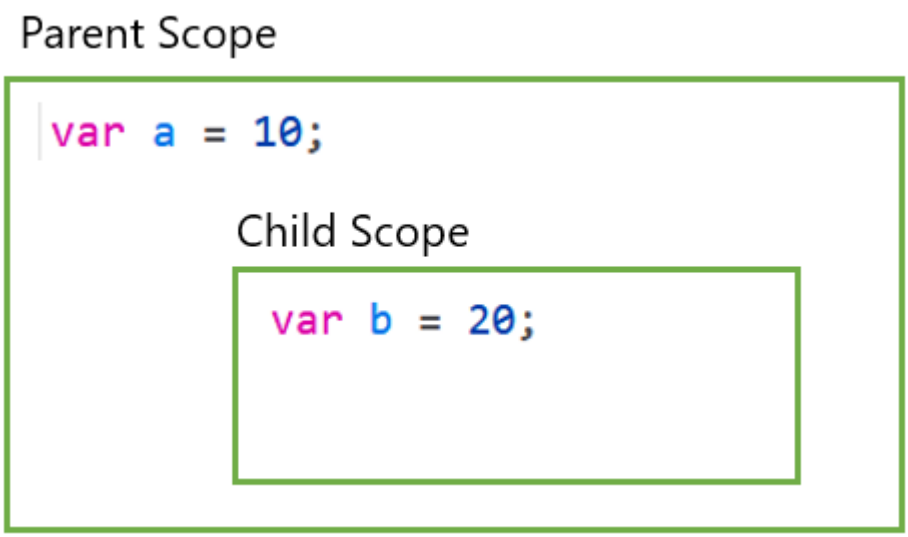
Example:

A Fish in an aquarium is having its scope inside of that aquarium only, when we bring the fish outside the aquarium then it will die. Means the scope of the fish is inside the aquarium only. So a variable declared inside of a block / scope is not accessible outside of that block or scope.

Let's discuss the concept of scope hierarchy

Scope Hierarchy

Scope Hierarchy means a chain of scopes which contains a parent scope and child scope.



Whatever the variable is declared inside parent scope are by default available to the child scope.

So 'x' is accessible in child scoped block.

But whatever the variable is declared in child scope is not available to parent scope by default.

So 'y' is not accessible outside of child scope and also inside parent scoped block.

Scopes Creation

In any other OOPS languages like C++ or Java, we will be creating scopes as

Example:

```
// Block Scoping
{
  var x = 20;
}

// Block Scoping

if(age <= 18){
   // some statements
}
```

So in Java, C++ just placing any code inside curly braces { } creates scope of a variable of block and outside of this block the variable is not accessible, this is called as block scoping.

But JavaScript does not supports block scoping and JavaScript supports Function Scoping. In JavaScript just by creating a block it will not be a separate scope creation. We can create a new scope using functions in JavaScript.

Function Scoping

In JavaScript, creation of scoping is not block level but it is a function level.

JavaScript does not encourage block scoping but in in JavaScript it is only Function Scoping.

Simply creating a block will not create a new scope in JavaScript.

Example:

UiBrains.com

```
var name = 'Naveen';
if(name === 'Naveen'){
    var designation = 'Software Engineer';
}
console.log(name); // Naveen
console.log(designation); //Software Engineer
```

Note: In the above example we declared a variable 'designation inside if block and still we can access that variable outside of the if block.

It clearly state that JavaScript doesn't support Block Scoping. JavaScript supports Function Scoping.

Now let's keep the same code inside a function.

Example:

```
var name = 'Naveen';

function allotDepartment() {
    if(name === 'Naveen'){
        var designation = 'Software Engineer';
    }
}

console.log(name); // Naveen
console.log(designation); // ReferenceError: designation is not defined
```

Now discuss few examples on Scoping Concepts

Example 1:

```
var top = 10;
function guess() {
    var inner = 20;
    console.log(inner);
}
guess(); // 20
```

In the above example we declared a variable 'inner' inside guess() function and we accessed that variable inside of that function only.

Example 2:

```
var top = 30;
var inner = 30;
function guess() {
    var inner = 40;
}
console.log(inner); // 30
```

Here in the above example we declare two inner variables one is outside of the function and one is inside a function. But we are accessing an inner variable outside of the function then obviously we cannot access the variable which is declared inside of the function. So we can access the variable declared outside of the function, because we are executing console.log outside of the function only.

IIFE (Immediately Invoked Function Expression)

We normally create variables outside of functions and they can be accessed anywhere of the program. These variables are called as global scoped variables.

But in JavaScript creating variables in global scope is not a good idea.

For any web application there may be a lot of JavaScript files will be executing on the browser. Then the variable created in global scope are stored in the same memory area of all the JavaScript files global scoped variables and it is accessible to all the other JavaScript files also. This kind of behavior causes very serious issues in JavaScript programming.

So placing everything in global scope is not at all recommended and it is not a good programming practice.

In order to avoid the global scoping, we can create those variable inside a function.

Example:

```
function addFn() {
    var a = 10;
    var b = 20;
    console.log(a + b);
}

addFn(); // 30
```

In order to avoid creating a, b variables globally we have created inside a function. But just by creating a function and adding variables will not get printed anything on the console. So we needs to call the function also.

But here there is a new problem arises, we have created variables inside the function and we are calling the function 'addFn()' from global scope.

Let's assume, if any other JavaScript file is also have the same function and we are calling that function. Now again the same problem with global scoping.

In order to avoid this problem we can use IIFE as follows,

```
(function() {
    var a = 10;
    var b = 20;
    console.log(a + b);
})();

// output : 30
```

Advantages of IIFE:

1) Avoids creating variables in global name space.
2) Functions doesn't have a name to collide with any other JavaScript files.
3) Avoid Global scoping of function call.

Read & Write Operations

This is not related database level read & write operations. This read and write operations are specific to declaration and definition of variables and reading them.

This concept is important to understand the behavior of variables in JavaScript.

Example:

```
var a = 10; // Write Operation
console.log(a); // Read Operation
```

In the first line, we are writing a value 10 to variable 'a'. This is called as read operation.

In the second line we are reading a value from 'a' so it is a read operation.

```
var a  = 10; // write operation
var y = 20; // write operation

var z = y; // Read & write operations

// reading from y and assigning to z
```

Example

```
function greet(wish) { // write operation
    console.log(wish); // read operation
}

greet('Good Morning');
```

Implications of Read & Write Operations

In any other OOPS languages in order to use a variable, first thing is we have to declare the variable. We can use a variable only after declaring it.

Here in JavaScript the variable declaration is purely depend on what are going to do with the variable.

```
var name;
console.log(name); //undefined
```

Let's access an undeclared variable,

```
console.log(department); // ReferenceError: department is not defined
```

Let's assign some value to undeclared variable,

```
isJSEasy = true; // No error
```

```
isJSEasy = true; // No error
console.log(isJSEasy); // true
```

NOTE: Write Operation is possible for declared and undeclared variables. But for Read operation it is possible only for declared variables.

Compilation & Interpretation

As most of us knows that JavaScript is an interpreted language. But technically it is Compiled and Interpreted language.

When compared to other programming languages like Java and C++, there is actually a compilation step happens and then creates an intermediate code and that will be executed in runtime.

But in JavaScript, there is no such intermediate code like Java or C++. But still in JavaScript there is no intermediate code will be generated and the compilation step happens inside the source itself and execute the same source code only.

Browser actually executes the source code as is, but this does not means that there is no compilation step.

There is a compilation step in JavaScript and which is different from all other languages.

When a browser sees the JavaScript, it does not execute it directly there are some certain step happens looks at the code and note down some of the things and then starts the execution. This happens within the fraction of seconds before the actual execution starts.

Compilation Phase

Compilation step mainly looks for variable declaration and function declaration only.

```
1  var a = 10;
2
3  var b = 20;
4
5  console.log(a + b);
```

Global
a
b

Compilation step looks for the 'var' and add them to the global scope, not the assignment phase.

1) It looks for 'var a' and adds it to a global scope.
2) As it does not care about the assignment phase.
3) It looks for 'var b' and adds it to global scope.

4) Console.log() is not care by compilation , so finally after compilation phase we have two variable inside global scope.

Example 2:

```
var a = 10;
function addFn() {
    var b = 20;
    var c = b;
    console.log(a + b);
}

addFn();
```

Global Scope
a
addFn

addFn() Scope
b
c

Note : In the above example 'addFn()' itself is a variable which contains a value function.

Example 3:

```
var myName = 'Naveen';
function greet(name) {
    console.log('Good Morning ' + name);
}
greet(myName);
```

Global Scope
myName
greet

greet() Scope
Name

Note: For any function parameters in JavaScript are like local variables to its function.

Interpretation Phase

Interpretation phases follows compilation phase and it uses the scope chain created by the Compilation phase.

The Interpretation or execution phase is takes care about variable assignment and function execution.

Example:

```
1  var myName = 'Naveen';
2  function greet(name) {
3      console.log('Good Morning ' + name);
4  }
5  greet(myName);
```

Steps:

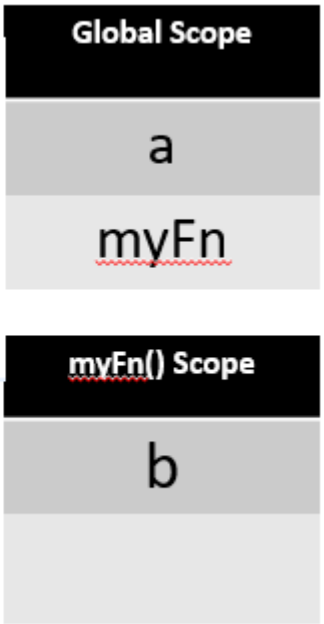
1) Interpreter takes in the string value 'Naveen' and assign to myName variable by following the global scope.
2) It Checks if the global scope is having a 'myName'. Once found it will assign to it.
3) Interpreter does not care about function declaration.
4) It goes to function execution 'greet(name)' as this line in global scope, it checks the variable 'greet' in the global scope.
5) It gets the greet variable in global scope and also myName which is actually there in global scope.
6) Now it goes to function definition, where in greet scope. It assigns and then goes to console.log() and prints the message to the console.

The Global Scope Problem

As of we use the variables which are already declared. What if we try to use the variable for assignment which is not even declared?

Example:

```
var a = 10;
function myFn() {
    var b = a;
    console.log(b);
    console.log(c); // RE "c is not defined"
}

myFn();
```



In the above example, the flow goes for compilation and execution, when the interpreter comes to the line

```
console.log(c);
```

It looks for the variable 'c' whether it is declared in myFn Scope,as myFn scope does not have the variable declared.

It goes to one level up and checks in the global scope, if it contains a declared variable 'c'. as the Global scope also doesn't contain a variable 'c'.

Now the variable is not declared anywhere else of the program. The decision will be taken by JavaScript Interpreter based on READ or WRITE operation.

As this is a READ operation and as we know the READ operation is not possible for non-declared variables, so it throws a runtime error.

Let's say instead of read operation we have a write operation.

```
var a = 10;
function myFn() {
    var b = a;
    console.log(b);
    c = 100;
}

myFn();
```

Again the same flow of compilation and interpretation.

And the interpreter comes to the line,

```
c = 100;
```

The interpreter looks for the variable declaration in the myFn Scope and not available and goes to one level up and looks for the variable in the global scope. The variable is still not declared in the global scope.

The decision made based on the type of operation. Since this is a 'write' operation. Write operation is possible for non-declared variables.

Now the interpreter go ahead and creates a variable in the **global** scope.

As this is one of the wired behavior of JavaScript, as we are using it in the myFn scope and JavaScript creates in the global scope.

Once a variable is declared in the global scope, this may accessible to some other JavaScript files and if they can use it and change it, the entire application may fail due to this nature.

So creating Global variables in any other languages is bad and it is even worst in JavaScript.

Note:

In order to avoid such situations we should always declare a variable with 'var' keyword before we starts using it.

UIBrains.com

Coding Exercises

Let's understand the below code, checks for the output.

```
var a = 10;
function outer() {
    var b = a;
    console.log(b);
    function inner() {
        var b = 20;
        var c = b;
        console.log(c);
    }
    inner();
}
outer();

output : 10 & 20
```

Global Scope
a
outer

outer Scope
b
inner

Inner Scope
b
c

Let's understand the below code, and guess the output

```
var a = 10;
function outer() {
    var b = a;
    console.log(b);
    function inner() {
        var c = b;
        console.log(c);
        var b = 20;
    }
    inner();
}
outer();

output : 10 & undefined
```

Global Scope
a
outer

outer Scope
b
inner

Inner Scope
c
b

In the above example, the compilation step looks for inner scope and variable c, b creates in the inner scope.

Now interpreter come into picture and goes to the line

```
var c = b;
```

And checks it contains a variable 'b'. As there is a variable in the inner scope it just declared any value to it. So it contains the value 'undefined' so the same value of 'b' will be assigned to the variable 'c'.

Now it prints 'undefined' to the console and finally assigns a value '20' to the variable 'b'.

Summary:

Let's summarize the above example in a simple way as follows,

```
1  console.log(a);
2  var a = 20;
```

As at line number 1. It is a read operation and at this time the variable is not declared. We may think like we get an error.

But if observe the code carefully, before interpretation executes the console.log() , there is a compilation step and which takes care about variable declaration. So before executing the console.log(), the variable has been declared already and assigned the value 'undefined' to it.

So it prints the value 'undefined' to the console and then the interpreter assigns the value '20' to 'a'.

Hoisting in JavaScript

Hoisting is a concept available in JavaScript and once we understand about compilation and interpretation steps, this will be to understand.

The concept of hoisting is that, no matter where we declare the variables, after compilation step and before interpretation step happens all the declarations will be hoisted to the top of the declared scope.

This concept is called Hoisting in JavaScript.

Example:

```
1  a = 10;
2  console.log(b);
3  c++;
4
5  var a;
6  var b;
7  var c;
```

In the process of compilation it scans the entire JavaScript file and it looks for variable declarations hoist or move to the top of the declared scope.

Once the compilation step is over then interpretation follows the scope chain along with the variables declarations and starts executing them.

Once all the variables are hoisted to the top of the program, then before interpretation step the code may looks as follows,

```
1  var a;
2  var b;
3  var c;
4
5  a = 10;
6  console.log(b);
7  c++;
```

Note: This hoisting process is not only applies to variables, it also applies to functions.

Example:

```
// function execution
sum();

// function definition
function sum() {
    // Logic to sum
}
```

Note: even though the function declaration is placed after the functions execution, in with the process of hoisting in JavaScript (in Compilation Phase) the definition of the function is hoisted to the top.

Note: The hoisting is not possible for function expressions.

Example:

```
greet();

var greet = function() {
   console.log("Hello Good Morning");
}

//Output : TypeError: greet is not a function
```

In the above code first compilation step takes care about the variable declaration, and it just creates a variable 'greet' in the global scope with 'undefined' as the value.

Now the interpreter comes to execute the function in line number 1, as this variable already contains a value as 'undefined' and interpreter can't be able to execute the 'greet()' as this is not a function at this point of time.

Due to this reason Hoisting is not possible for Function Expressions and the hoisting works for function declaration.

Note: While using function expressions in a JavaScript codebase, make sure you write a function expression and then execute that function expression.

Using Strict Mode:

As we discussed earlier that read operation is not possible for non-declared variables and only write operation is possible for non-declared variables.

If we are trying to apply the write operation on non-declared variables, it is allowed in JavaScript but the JavaScript itself creates that variable in '**Global**' scope instead of its declared scope.

In any programming language declaring any variables in the global scope is bad, it is even worst in JavaScript programming.

```
1  var myName = 'Naveen';
2
3  // some serious logic
4
5  myname = 'Test';
6
7  // some serious logic
```

Let's understand the above example, at line number 1 we have created a variable '**myName'** and assigned a value, after some serious code, we have assigned a value to some other variable called '**myname**' instead of '**myName**' and which is not declared at all. As in JavaScript the variable names are case sensitive, and both are two different variables.

As the variable '**myname**' is not declared anywhere else in the program, and as it is a write operation and also write operation is possible for non-declared variables so JavaScript creates this variable in the global scope (check the window object on the browser)

```
▶ gSnippetsMap: Map { "appData.defaultBrowser" → false, "appData.fxa
▶ getBlockList: function getBlockList()
  myName: "Naveen"
  myname: "Test"
▶ popFromBlockList: function popFromBlockList()
▶ removeMessageListener: function removeMessageListener()
▶ sendAsyncMessage: function sendAsyncMessage()
```

Due to the global variables are bad in JavaScript and we haven't expected the behavior of JavaScript creating a variable in the global scope.

In order to avoid this, from ES5 version we got a new feature called '**Strict Mode**'.

If we run any JavaScript in Strict Mode, It warns the non-declared variables as follows,

```
"use strict";
var myName = 'Naveen';
// some serious logic
myname = 'Test';

// ReferenceError: assignment to undeclared variable myname
```

'**Note**: We can use this Strict Mode to the entire JavaScript to run it in the strict mode just by adding this line to the top of any JavaScript File.

```
1  "use strict";
```

We can even execute only a function in strict mode, by adding the above line inside a function.

```
function greet(){
  "use strict";
  // Some serious logic
}
```

Note: We should always run the JavaScript in Strict Mode to avoid such errors as mentioned above.

Closures

This is one of the most important concept in JavaScript along with Scopes. This allows the programmers to write better, creative and expressive code.

A Closure is an inner function that has access to the outer (enclosing) function's variables scope chain.

The Closure is having three scope chains as follows,

1) It have access to its own scope (the variables defined in its curly braces).
2) It has access to the outer functions variables.
3) It has access to the Global Variables.

Example:

```
// Closure Example 1
var a = 10;
function outer() {
    var b = 20;
    function  inner() {
        console.log(a);
        console.log(b);
    }
    inner();
}
outer();

// Output : 10 20
```

In the above example variable 'a' is created in the global scope and variable 'b' is created in the outer scope.

We are executing the 'outer()' function in the global scope, it goes to outer() function definition and creates a variable 'b' and also executes the inner() function.

While executing the inner(), there is no variable 'a' in the inner() and outer() function, so the variable gets from the global scope and prints it on the console. And trying to access the variable 'b' in the inner() and it is not there in the inner() and then the control goes one level up and gets the variable 'b' and prints it on the console.

```
1   // Closures Example 2
2   var a = 10;
3   function outer() {
4       var b = 20;
5       var inner = function () {
6           console.log(a);
7           console.log(b);
8       };
9       return inner;
10  }
11
12  var innerFn = outer();
13  innerFn();
14
15  // Output : 10 20
```

In the above Example, instead of a normal function inner(), we have created a function expression, this is an isolated function inside outer().

At line number 12, we are executing the function outer(), here the variable 'b' will created in the outer() scope and the isolated function will be returned and which is assigned to 'innerFn'.

At Line number 13, we are executing an isolated function and here there is not outer involved. At this place the variable 'a' is declared in the global scope and it is available to 'innerFn' , but the scope of variable 'b' is to outer() only and how does the variable 'b' is accessible here? (this is the concept of closure).

The Whole concept of closures is, at the time a function creation it remembers its scope chain and if we execute that function in a totally different context, it still remembers its scope chain. This is the concept of closure.

UiBrains.com

```
1   // Closure Example 3
2
3   var a = 10;
4   function outer() {
5       var b = 20;
6       var inner = function () {
7           console.log(a);
8           console.log(b);
9       };
10      return inner;
11  }
12
13  function processEngine(fName) {
14      fName();
15  }
16
17  var innerFn = outer();
18  processEngine(innerFn);
19
20  // Output : 10 20
```

In the above Example, at line number 17 we got an isolated function 'innerFn', this variable we are passing to another function called 'processEngine()' this is even inside a different context.

Here in line number 18, we are executing the processEngine() function, due to the concept of closures it still remembers its scope chain and prints the values 10 , 20 to the console.

```
1   // Closure Example 3
2
3   var a = 10;
4
5   function outer() {
6       var b = 20;
7       var inner = function () {
8           a++;
9           b++;
10          console.log(a);
11          console.log(b);
12      };
13      return inner;
14  }
15
16  var innerFn1 = outer();
17  innerFn1(); // 11 21
18  var innerFn2 = outer();
19  innerFn2(); // 12 21
```

In the above example, whenever we execute a function, then a new copy of variables will created. And there will only one copy of global variable.

Due to the concept of closures and above information it prints the values as specified in the example.

The Module Pattern

The Module pattern is one of the most commonly used design pattern in JavaScript to create Encapsulation for our code.

Normally in any other language like Java, we may create an encapsulation using 'private' keyword. The variables which are declared with 'private' keywords are not accessible to outside. Only through setter or getter method we will allow the access.

As in JavaScript there is no concept of 'private' keywords, all the properties / variables which are declared are by default 'public'.

By using the concept of Closures we can implement the Encapsulation to our JavaScript Code.

Here if we create any variables inside an objects, which accessible inside the functions inside an object as follows,

```
var employee = {
    firstName : "Naveen",
    lastName : "Saggam",
    getFirstName :function () {
        return this.firstName;
    },
    getLastName : function () {
        return this.lastName;
    }
};

console.log(employee.firstName); // Naveen
console.log(employee.getFirstName()); // Naveen
```

In the above example, we can access the first Name directly and also by using the getFirstName() function.

As per encapsulation concept we shouldn't access the firstName and lastName properties directly, we should be able to access them only by using the function names.

We can achieve this by the concept called **closures.** As follows,

```
function employee() {
    var firstName = "Naveen";
    var lastName = "Saggam";

    var empObj = {
        getFirstName : function() {
            return firstName;
        },
        getLastName :function() {
            return lastName;
        }
    };
    return empObj;
}

var employee = employee();

console.log(employee.firstName); // undefined
console.log(employee.getFirstName()); // Naveen
```

The above example denotes the Module Pattern in JavaScript where we implemented the concept of Encapsulation.

Objects in depth

Here we understand the concept of Constructor functions in JavaScript. The usage of constructors is one of the best way of creating objects.

First we understand the creation of objects and also observe how we can do this in an easy way using constructor functions.

We can create an object in JavaScript as follows,

```
// Simple Objects creation and access in JS
var Obj = {};
Obj.firstName = "Naveen";
Obj.lastName = "Saggam";

console.log(Obj);
console.log(Obj.firstName);
```

Here we have created an empty object and added all the properties to it after creation of an empty object.

Let's create an employee object using the above approach,

```
// Creation of Employees Objects with raw data
var employee1 = {};
employee1.firstName = "John";
employee1.lastName = "Kennedy";
employee1.gender = "M";
employee1.designation = "Regional Manager";

console.log(employee1);
```

Let's create another employee object,

```
var employee2 = {};
employee1.firstName = "John";
employee1.lastName = "Cena";
employee1.gender = "M";
employee1.designation = "Delivery Manager";

console.log(employee2);
```

In order to create an employee management system and create a 100 employee objects, we may have to repeat the above steps for all the 100 employees.

Instead of this we can create a function do create an object and stuff all the properties and returns that object.

Example:

```
// Creation of a function to produce Employee Object
function createEmployee(firstName,lastName,gender,designation) {
    var employeeObj = {};
    employeeObj.firstName = firstName;
    employeeObj.lastName = lastName;
    employeeObj.gender = gender;
    employeeObj.designation = designation;
    return employeeObj;
}

var employee3 = createEmployee("Ram","Rajan",38,"Sales Representative");
console.log(employee3);
```

By using the above function we can create any number of Employee Objects using only one function.

This kind of Objects creation pretty common in JavaScript, for this approach we may use a new function called 'constructor function'.

In the above function, we created an empty object and added all the properties to that object and returns that object. So creation of an empty object and returning of that object is common for all the objects creation inside functions. This can be done automatically by JavaScript using **Constructing Functions**.

Example:

```
// JS Constructors Example
function CreateEmployeeConstructor(firstName,lastName,gender,designation) {
    this.firstName = firstName;
    this.lastName = lastName;
    this.gender = gender;
    this.designation = designation;
}
var employee4 = new CreateEmployeeConstructor("Ram","Rajan",38,"Sales Representative");
console.log(employee4);
```

The above example denotes the JavaScript's Constructor function, we need to add the properties to **this** object (which is provided by JavaScript automatically) and also while executing the constructor function we must call the function using **new** operator.

While executing a constructor function using **new** keyword informs JavaScript to execute a function in a Constructor way.

Interview Questions on JavaScript

1) What is JavaScript
2) What is the difference between Java & JavaScript
3) What are the first class functions in JavaScript?
4) Explain the different datatypes of JavaScript.
5) What is the difference between undefined and null.
6) How many ways we can print the logs in JavaScript and explain.
7) What are the different ways of using JavaScript and which approach is the best and why?
8) Explain about String Concatenation operator in JavaScript.
9) What is the use of 'typeof' operator in JavaScript?
10) What is the difference between = , == , === operators in JavaScript
11) Print the values from 1 – 10 using for loop, while, and do while loops.
12) What is the use of 'break' statement in Switch statements what happens if we don't use break statements in switch.

13) Explain about objects in JavaScript and the ways of creation of objects in JavaScript.
14) What is the difference between DOT notation and [] notation and when to use what.
15) Explain about nested Objects creation, usage and access of properties of nested objects.
16) What is an Array and usage of array?
17) What is the difference between shift() and unshift() functions of an array.
18) What is the difference between push() and pop() in an array.
19) What is the difference between splice() and slice() in array.
20) Usage of join() function in an array of JavaScript.
21) What is the JavaScript function and execution?
22) Is JavaScript function accepts less / more number of parameters as defined in the function definition?
23) Why function overloading is not possible in JavaScript?
24) How to call a function with return values.
25) What is the difference between function expression and anonymous function expression?
26) Can we pass a function itself is an arguments to another function? Explain with an example.
27) Explain about functions inside an object and how to access the properties of an object inside the function.
28) Generate some Random numbers from 1 – 1000 using Math Object.
29) Write a code to display a full day using Date Object in JavaScript.
30) Write a code to display a digital clock in Indian Time Zone.
31) What is DOM manipulation in JavaScript.
32) What is the difference between DOM and BOM in JavaScript.
33) Explain some of the BOM properties in JavaScript.
34) Explain about few DOM properties and methods.
35) What is the difference between document.querySelector() and document.querySelectorAll() methods.
36) What is an Event in JavaScript and what is the various DOM Events.
37) Explain the process of DOM Event Handling in JavaScript.
38) Explain about the usage of DOM Event listeners in JavaScript.
39) What is Scope in JavaScript?
40) What is the difference between block scoping and Function Scoping in JavaScript?

uibrains.com

41) Explain about the Scope hierarchy in JavaScript.
42) How to create scopes in JavaScript?
43) Why JavaScript supports only function scoping.
44) Explain about IIFE in JavaScript.
45) What are the advantages of IIFE in JavaScript?
46) Explain about Read & Write Operations in JavaScript.
47) What are the implications of Read & Write Operations in JavaScript?
48) Explain about the compilation phase of JavaScript.
49) Explain about the Interpretation Phase of JavaScript.
50) What is the Global scoping problem in JavaScript?
51) What is Hoisting in JavaScript?
52) Explain the Hosing concept for variables and functions.
53) Why Hoisting is not possible for function Expressions?
54) What is the usage of Strict Mode in JavaScript?
55) Explain the Closures concept in JavaScript.
56) Explain about the Module Pattern in JavaScript
57) What is the difference between the normal function and Constructor function in JavaScript?
58) Write a logic for this in JavaScript.
    The parameter weekday is true if it is a weekday, and the parameter vacation is true if we are on vacation. We sleep in if it is not a weekday or we're on vacation. Return true if we sleep in.

    sleepIn(false, false) → true
    sleepIn(true, false) → false
    sleepIn(false, true) → true

59) Write a logic for this in JavaScript
    We have two monkeys, a and b, and the parameters aSmile and bSmile indicate if each is smiling. We are in trouble if they are both smiling or if neither of them is smiling. Return true if we are in trouble.

    monkeyTrouble(true, true) → true
    monkeyTrouble(false, false) → true
    monkeyTrouble(true, false) → false

60) Write a logic for this in JavaScript

Given two int values, return their sum. Unless the two values are the same, then return double their sum.

sumDouble(1, 2) → 3
sumDouble(3, 2) → 5
sumDouble(2, 2) → 8

61) Write a logic for this in JavaScript
Given an int n, return the absolute difference between n and 21, except return double the absolute difference if n is over 21.

diff21(19) → 2
diff21(10) → 11
diff21(21) → 0

62) Write a logic for this in JavaScript
We have a loud talking parrot. The "hour" parameter is the current hour time in the range 0..23. We are in trouble if the parrot is talking and the hour is before 7 or after 20. Return true if we are in trouble.

parrotTrouble(true, 6) → true
parrotTrouble(true, 7) → false
parrotTrouble(false, 6) → false

63) Write a logic for this in JavaScript
Given a string and a non-negative int n, return a larger string that is n copies of the original string.

stringTimes("Hi", 2) → "HiHi"
stringTimes("Hi", 3) → "HiHiHi"
stringTimes("Hi", 1) → "Hi"

64) Write a logic for this in JavaScript
Given a string and a non-negative int n, we'll say that the front of the string is the first 3 chars, or whatever is there if the string is less than length 3. Return n copies of the front;

frontTimes("Chocolate", 2) → "ChoCho"
frontTimes("Chocolate", 3) → "ChoChoCho"
frontTimes("Abc", 3) → "AbcAbcAbc"

65) Write a logic for this in JavaScript
Count the number of "xx" in the given string. We'll say that overlapping is allowed, so "xxx" contains 2 "xx".

countXX("abcxx") → 1
countXX("xxx") → 2
countXX("xxxx") → 3

66) Write a logic for this in JavaScript
Given a string, return the count of the number of times that a substring length 2 appears in the string and also as the last 2 chars of the string, so "hixxxhi" yields 1 (we won't count the end substring).

last2("hixxhi") → 1
last2("xaxxaxaxx") → 1
last2("axxxaaxx") → 2

67) Write a logic for this in JavaScript
Given two strings, append them together (known as "concatenation") and return the result. However, if the concatenation creates a double-char, then omit one of the chars, so "abc" and "cat" yields "abcat".

conCat("abc", "cat") → "abcat"
conCat("dog", "cat") → "dogcat"
conCat("abc", "") → "abc"

68) Write a logic for this in JavaScript
Given two strings, append them together (known as "concatenation") and return the result. However, if the strings are different lengths, omit chars from the longer string so it is the same length as the shorter string. So "Hello" and "Hi" yield "loHi". The strings may be any length.

minCat("Hello", "Hi") → "loHi"
minCat("Hello", "java") → "ellojava"
minCat("java", "Hello") → "javaello"

69) Write a logic for this in JavaScript
Given 2 int arrays, each length 2, return a new array length 4 containing all their elements.

plusTwo([1, 2], [3, 4]) → [1, 2, 3, 4]
plusTwo([4, 4], [2, 2]) → [4, 4, 2, 2]
plusTwo([9, 2], [3, 4]) → [9, 2, 3, 4]

70) Write a logic for this JavaScript
Given 2 ints, a and b, return their sum. However, "teen" values in the range 13..19 inclusive, are extra lucky. So if either value is a teen, just return 19.

teenSum(3, 4) → 7
teenSum(10, 13) → 19
teenSum(13, 2) → 19

UiBrains.com

TypeScript

Introduction:

Typescript is a superset of JavaScript best transpiler that compiles the Object Oriented code to plain JavaScript. This is a pure Object Oriented Language with Classes and Interfaces. This is an Extension of JavaScript where we can write OOPS code and that compiles back to pure / Core JavaScript which runs on any browser , any OS and any Device.

TypeScript is developed and Maintained by Microsoft and is an Open Source Project.

The Popular Front End Framework Angular 5 is also built on TypeScript.

The Current version is 3.0.3 released Sep, 2018.

Typescript is one of the best transpiler as of today which supports almost all the modern features of ECMA Script and in future also it will supports all the new upcoming features of ECMA Script.

We will write code in TypeScript and compiles it back to core JavaScript of any version.

As JavaScript is a weakly typed language where in TypeScript is a Strongly Typed Language, in order to avoid unnecessary errors caused by JavaScript at Runtime.

TypeScript Summary:

Type	Details
Developed by	**Microsoft**
License	**Open Source**
Current Version	**3.0.3 Sep,2018**
Official Website	https://www.typescriptlang.org/
Practice Online	https://www.typescriptlang.org/play/index.html
Known as	**Best Transpiler for JavaScript**
Compatible with	**Any Browser , Any OS , Any Device**

ECMA Script

JavaScript is also called as ECMA Script. ECMA Stands for European Computers Manufacturers Association.

ECMA is the standard body which releases the different versions of ECMA Script / JavaScript. The current version of ECMA Script is **ECMA Script-2017** (8th Edition).

This ECMA Script language features can be used by all the browsers, to implement the support for running various versions of JavaScript/ECMA Script.

As ECMA releases the new features, which are not automatically available by all the browsers and most browsers supported version of ECMA Script is **ES-5** version.

This ECMA Script 5 is the version, which is used by almost all the browsers supports as today.

Here we needs to understand if ECMA releases a new feature / standard, however it does not mean that every feature of the standard body ECMA is the standard and immediately available in all the browsers.

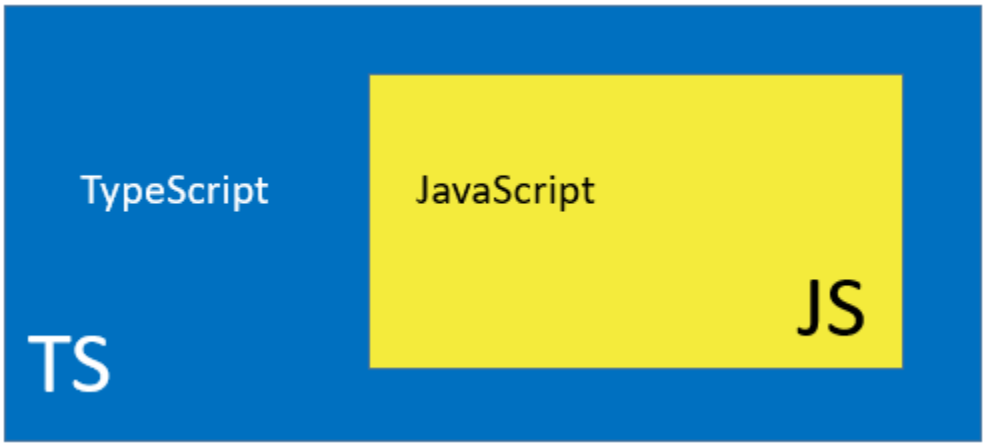
In reality some implementations / features are already available in all the browsers and which are not standardized by ECMA and also the features and some implementations are standardized by ECMA which are not available in all browser.

Now we understood that there is a gap between the ECMA Script versions used by all the modern browsers and the version which are released by ECMA.

As a JavaScript developer may get frustrate because there are lot of new features coming in but there is no actual browser support for those features in order to leverage those features.

This is the place where Transpiler comes into picture to fill the gap, where the transpiler compiles and transforms from one standard to another standard.

So TypeScript is one of the best transpiler as of today and we can leverage all the features of ECMA script and even future versions of ECMA Script and transpiles to ES-5 and which is fully supported by all the browsers as of today.



There are many Transpilers available in the market along with TypeScript is Babel Script. TypeScript is more popular and even the Angular Framework uses TypeScript.

TypeScript supports all the features of ES6 and also the future versions of ECMAScript and compiles them to any chosen version of JavaScript like ES5.

Differences between TYPESCRIPT and JAVASCRIPT

JAVASCRIPT	TypeScript
JAVASCRIPT is a Dynamic Typed Language	TS is a Static Typed Language
This is most forgiving language and weakly typed language	This is particular about typing and it is a strongly typed language like Java
As this is not a Strict typing, it leads to many unexpected runtime errors during development time	As this is a Strong typing, it avoids almost all unexpected runtime errors during development time, especially unexpected errors seen in JAVASCRIPT
It applies typing in Dynamic runtime and which leads to many runtime errors	It applies type in the development time itself and avoids runtime errors
JAVASCRIPT is great for Web Browser Object Model	It promotes stability and maintainability

NOTE:

JavaScript is a dynamic typing language means we can provide the typing information for any variables at runtime. But for TypeScript we can provide the typing information only at the time of declaration of variables. This is called static typing.

As JavaScript is dynamic typing language, it is best suitable for BOM (browser object model) and with TypeScript adds static typing to the JavaScript to add more stability and more maintainability of the program and avoids unexpected runtime errors for JavaScript.

Instead of writing direct JavaScript, if we use TypeScript to generate JavaScript which compiles and gives the most stable and most maintainable JavaScript and most powerful JavaScript.

So this is the reason why Angular JS is completely redesigned to use TypeScript and renamed to Angular.

TYPESCRIPT Editors

We can use any text editor to work with TypeScript as follows,

1. Any text editor + TS compiler (command prompt)
2. Atom Text Editor by Github - open source

3. Visual Studio Code by Microsoft - open source
4. Brackets Editor by Adobe Systems - open source
5. Sublime Text by Sublime – Semi licensed
6. WebStorm by JetBrains – Fully Licensed

Installing TypeScript

We can install the Type using the Node JS' Node Package Manager (npm).

1. Install Node JS
2. Check the versions of **node** and **npm**
3. Install "typescript" using npm

INSTALL

```
npm install -g typescript
```

4. Check the version of TypeScript

```
tsc -v
```

5. Compile the TypeScript using **tsc**

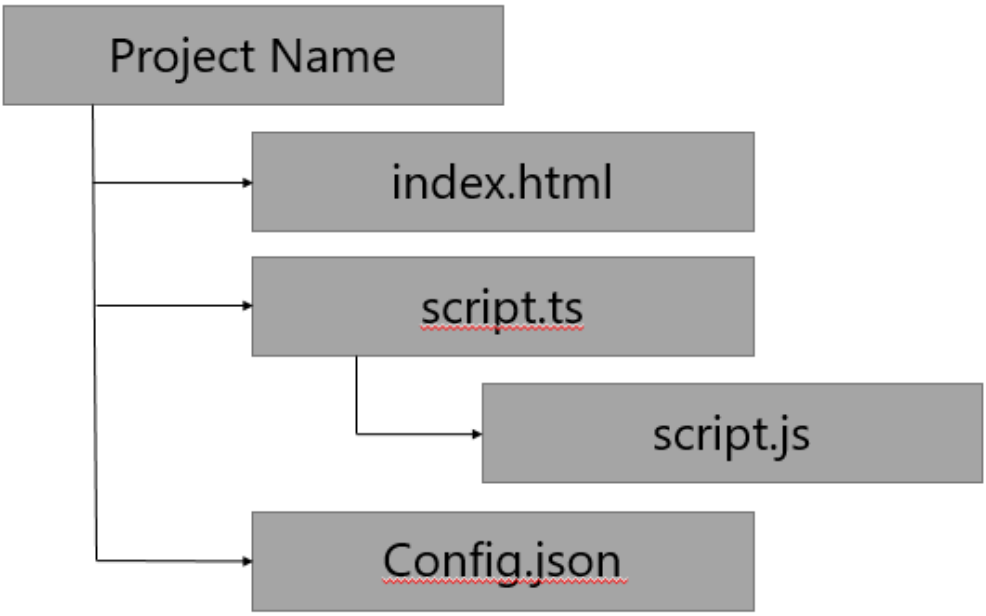
COMPILE

```
tsc helloworld.ts
```

UiBrains.com

Project Structure of TypeScript

We must follow the below project structure for the entire course.



ECMA Script 6 Features:

TypeScript leverages to use all the ECMA script features and compile them to 'ES5' plain JavaScript to work on any modern browser.

Here we use all the 'ES6, features in TypeScript and compile them to 'ES5' JavaScript code and execute them in any browser.

Now let's understand the 'ES6' features and their usage.

Default Parameters:

This feature is to provide some default values to optional parameters, if any of the parameter is not supplied, then the program will be using these default parameters.

```
function countdownCompleted(final,initial,interval){
    let output = "";
    let current  = initial;
    while(current <= final){
        output += " " + current;
        console.log(current);
        current += interval;
    }
    document.getElementById('display').innerHTML = output;
}

countdownCompleted(20);
```

In the above example the function takes in the parameters final, initial, interval.

Let's make "initial" and "interval" are optional parameters, we can use in the following way:

```
function countdownCompleted(final,initial=0,interval=2){
    let output = "";
    let current  = initial;
    while(current <= final){
        output += " " + current;
        console.log(current);
        current += interval;
    }
    document.getElementById('display').innerHTML = output;
}

countdownCompleted(20);
```

In the above function, we have provide a default parameters for **initial** and **interval** values,

If that values are not provided, then these default values will take into consideration.

Template Strings

Normally we use String Concatenation operator to combine two or more strings. This approach is more difficult if mix the HTML tags with normal Strings. We resolve this by using Template Strings.

In this template string, we use back tick (`) symbol to combine any kind of strings literals or HTML code or any JavaScript code instead of using string concatenation operator.

By using String Concatenation operator we can mix the HTML code with normal Strings as follows,

```
var stringConcat = "<div>" + "<h1> Name : " + employee.name + "</h1>" +
                             "<h1> Age : " + employee.age + " </h1>" +
                             "<h1> Desg : " + employee.desg + "</h1>" +
                     "</div>";
```

We can replace this by using Template String with back tick (`) operator as follows,

```
var templateString = `<div>
                          <h1>Name : ${employee.name}</h1>
                          <h1>Age : ${employee.age}</h1>
                          <h1>Desg : ${employee.desg}</h1>
                        </div>`;
```

Let and Const

Let is one of the **ECMA Script6** feature which is used to declare variables in TypeScript same as the '**var**' keyword in JavaScript.

The problem with the **var** keyword is, once we create a variable inside if block or for loop, we can still access that variable outside if block or for loop. We can avoid access of the variables which are declared inside if block or for loop using **let** keyword.

```
for(var i=0; i<= 10; i++){

}
var output = " The value of 'i' is : " + i;

//Output : The value of 'i' is : 11
```

We can avoid the access any variable declared inside any block is by using **let** keyword as follows,

```
for(let j=0; j<=10; j++){

}
var output = " The value of 'j' is : " + j;

//Output : ReferenceError: j is not defined
```

Const keyword is used to declare some final variables, once a variable is declared with **const**, we can't able to change the value of it and also it avoid access of that variable outside the declared block.

```
const  MONTH_NAME = 'January';
output = "The Month name is : " + MONTH_NAME;
```

Once a variable is declare using **const**, we can't change the value of it. If we are trying to change the value of it, we may get the Compile Time error as follows,

```
const  MONTH_NAME = 'January';
output = "The Month name is : " + MONTH_NAME;

MONTH_NAME = "February";

// Output : TypeError: invalid assignment to const `MONTH_NAME'
```

For …of loops

Normally we use **for…in loop** to loop through an array and we will get the index and with the index we can get the actual value of an array as follows,

This is available in ES5 version only.

```
1 let myArray = ['html','javascript','css','bootstrap'];
2
3 let output = "";
4 for(let index in myArray){
5     let value = myArray[index];
6     console.log(value);
7     output += value + " , ";
8 }
9 document.getElementById('display').innerHTML = output;
```

From ES6 version onwards, ECMA has provided a new concept called **for…of loop**.

By using this for…of loop, we can directly get the actual value of an array instead of getting an index of an array as follows,

```
1 let myArray = ['html','javascript','css','bootstrap'];
2 let output = "";
3 for(let value of myArray){
4     console.log(value);
5     output += value + " , ";
6 }
7 document.getElementById('display').innerHTML = output;
8
9 // output : html , javascript , css , bootstrap
```

Lambdas: [Arrow function]

A lambda expression is an anonymous functions that replaces the normal function definition with shortcut arrow functions.

This is one of the functional programming feature, we can use this feature to replace the function parameters with arrow functions.

Let's understand a function Expression call without using Arrow function as follows,

```
1 let greet = function():string{
2     return "Good Morning";
3 };
4
5 let output = greet();
6 console.log(output);
```

Now let's replace the function call at line number 5 using Arrow function as follows,

```
let greet = function():string{
    return "Good Morning";
};

let output = () => "Good Morning";
console.log(output);
```

In the above example, instead of creating and executing the function greet(); we can directly use the arrow function.

```
() => "Good Morning";
```

Let's understand how to replace an anonymous function expression that takes in few arguments with lambda / Arrow functions as follows,

```
1 let sum = function(a,b){
2   return a + b;
3 };
4
5 let sumArrow = (a,b) => a + b;
6 let output = "The Sum is : " + sumArrow(10,20);
7 console.log(output); // The Sum is : 30
```

In the above example, the sum function we replaced with an arrow function in the line number 5. This is one of the shortcut method to replace function expressions using Arrow Function.

Let's have an another function which takes an array and prints the length of passed in array using arrow functions as follows,

```
1 let len = function (array:Array<number>) {
2   return array.length;
3 };
4
5 let lenArrow = (array) => array.length;
6 let myArray = [10,20,30,40,50];
7 let output = lenArrow(myArray);
```

In the above example, we replaced the len() with an arrow function in the line number 5.

We can also use Arrow functions replace the map() function of an array as follows,

```
1  let movies = ['Baahubali' , 'Dangal' , 'Drushyam'];
2
3  // Normal Function
4  let movieLength = movies.map(function (movie) {
5          return movie.length;
6      }
7  );
8  console.log(movieLength); // [9, 6, 8]
9
10 // Lambda Expressions
11 movieLength = movies.map( movie => movie.length);
12
13 let output = movieLength.join(" - ");
14 console.log(movieLength); // [9, 6, 8]
```

In the above example, we are printing the length of each of the element and creating an array using movies.map() function.

We can replace the parameter of movies.map() using Lambda expressions or Arrow Functions as specified in the line number 11.

We can also mix the default parameters with Arrow functions as follows,

```
1  let greet = function(name:string='Williams',age:number=35):void{
2      let output:string = "Hello " + name + " You are " + age + " yrs Old!";
3      console.log(output);
4  };
5
6  greet(); //Hello Williams You are 35 yrs Old!
7
8  let greetArrow = (name:string = 'Williams' , age:number = 35) =>
9                   ("Hello " + name + " You are " + age + " yrs Old!");
10
11 let output = greetArrow('John',45);
12 console.log(output); //Hello John You are 45 yrs Old!
```

Destructing

This is one of the powerful feature in ES6 in which instead of adding a single value to a variable from an array, we can do it all by once.

Here Instead of fetching each and every value from an array and adding to each variable created, we use the shortcut notation called **Destructing**.

We normally use the following way to access each element from an array as follows,

```
1  let array:any = [10001,'naveen','TechLead'];
2
3  function addVals(array){
4      let id = array[0];
5      let myName = array[1];
6      let designation = array[2];
7      console.log("Id : " + id + "My Name : " + myName + " Designation : " + designation);
8  }
9
10 addVals(array);
```

In shortcut we can achieve this by using the concept called **destructing** as follows,

```
1  let array:any = [10001,'naveen','TechLead'];
2
3  function addVals(array){
4      let [id,myName,designation] = array; // Destructing
5      let output = `Id : ${id} My Name : ${myName} Designation : ${designation}`;
6      console.log(output);
7  }
8
9  addVals(array);
```

In the above example, line number 4, we use the concept called **destructing**. At line number 5, we use the concept called **Template String** (back tick operator) for combining various stings and values.

We can also use the concept called destructing to exchange the values between two variables.

Normally to exchange values between two variables, we should require one temp variable as follows,

```
1  // To exchange the values between a & b
2  let a = 10;
3  let b = 20;
4  console.log(a + " , "+ b); // 10 , 20
5
6  // create a temp variable
7  let temp = b;
8  b = a;
9  a = temp;
10 console.log(a + " , "+ b); // 20 , 10
```

In a simple way of exchange the values between two variables is by using the concept called destructing as follows,

```
1  //To Exchange the values Using Destructing ES6
2  let a:number = 10;
3  let b:number = 20;
4
5  [ b ,a ] = [a , b];
6
7  let output = `a : ${a} , b: ${b}`;
8  console.log(output);
```

The Destructing concept even works for Objects as follows,

```
// Using Destrucing ES6 for Objects
let employee = {
    empName : 'John',
    age : 40,
    designation : 'Manager'
};

let { empName , age , designation } = employee; // using destructing

let output:string = `Emp Name : ${empName} Age : ${age} Designation : ${designation}`;
console.log(output);
```

The Spread Operator

The spread operator is allows an expression to be expanded in places where multiple elements / variables / arguments are expected.

If any function is expected zero or more number of arguments, we can use the spread operator to pass in the complete array to is.

```
function Employee(){

    let values = [];
    for(let i=0; i<arguments.length;i++){
        values[i] = arguments[i];
    }
    return values;
}

const employee1 = [1001 , 'Naveen','TechLead'];
let output =  Employee(...employee1);
console.log(output); // 1001 Naveen Techlead
```

In the above example we can pass in the complete array as a parameter to the Employee() function. Here the complete array will be expanded and pass it as an argument to the above function.

We can use the spread operator to insert an array in the middle of another array using spread operator as follows

```
let array1 = [3,4];
let array2 = [1,2,...array1,5,6];
let output = "The array2 values are : " + array2.join();
console.log(output); // 1,2,3,4,5,6
```

In the above example we have inserted the elements of an array1 into the middle of array2 using spread operator.

We can also use the spread operator to create a copy of an array and assign to another using spread operator, same like slice() function of arrays.

```
let array1 = [10,20,30,40,50];
let array2 = [...array1];
let output = array2.join(",");
console.log(output); // 10,20,30,40,50
```

In the above example, we created a new array with the same copy of array1 using the spread operator.

We can also use spread operator to concatenate two or more arrays as follows,

```
let array1 = [10,20,30,40];
let array2 = [50,60,70,80];

// Without using spread operator
//array1 = array1.concat(array2);
console.log(array1); // 10,20,30,40,50,60,70,80

// Using the spread operator
array1 = [...array1,...array2];
console.log(array1); // 10,20,30,40,50,60,70,80
```

We can also use Spread operator in conjunction with Math Operations as follows,

```
let myArray = [9,20,45,13];
let minVal = " The min value is : " + Math.min(...myArray);
console.log(minVal); // The min value is : 9
```

TypeScript Data Types

TypeScript data type are used to specify the type of data we pass to any variables. In JavaScript we won't specify the type of data at compile time, at runtime the assigned data will be identified by the JavaScript. By using TypeScript we can add static typing to the defined variables.

In JavaScript we can re-assign the values to any variables defined. This may cause unexpected runtime errors at runtime, in order to avoid these errors by using TypeScript we can add static typing to JavaScript using various TypeScript data types.

TypeScript supports the following data types.

Number

In order to specify any number to a variable, we use the data type number in TypeScript.

Let's understand the creation of variables in TypeScript as follows,

```
let num1: number = 10;
let num2: number = 20;
let sum:number = num1 + num2;
console.log(`The Sum of num1 , num2 is : ${sum}`); //The Sum of num1 , num2 is 30
```

In the above example, we declared two numbers and the sum of two variables assigned to a new variable sum and printed them on the console using Template String of ES6 version.

Note: Once we declare any variable of one type and try to re-assign them to another type, we may get the compile time error in TypeScript.

```
let num1: number = 10;
let num2: number = 20;

// re-assign with a string
num1 = 'test'; // CE: 'test' is not assignable type 'number'.
```

String

This is one of the commonly used data type for textual data of any length.

We use String data type to assign any textual data to any variable of any length. We can even assign a single character to a variable using this string data type. In TypeScript once a declare any variable with string data type, we cannot re-assign that variable with any other type of value to it. If we try to assign any other type of value to the string variable we will get a compile time error.

```
let greetMsg:string = "Good Morning";
let empName : string = "Naveen";
let output: string = `${greetMsg} ${empName}`;
console.log(output); // Good Morning Naveen
```

Boolean

We use Boolean data type to assign true or false value to a variable.

Once we declare any variable with Boolean data type, we can't assign any other type of value to the same variable.

```
let isTSEasy : boolean = true;
let output : string = `The value of isTSEasy is : ${isTSEasy}`;
console.log(output); // The value of isTSEasy is : true
```

Array

We use Array data type is to assign an array to a variable. Normally in JavaScript we can add any type of elements to an array, but in TypeScript we can restrict the type of elements can be used to in an array.

```
let uiTechnologies:Array<string> = ['HTML','CSS','JavaScript','Bootstrap','Angular JS'];
let output:string = `UI Technologies : ${uiTechnologies.join(",")}`;
console.log(output);//UI Technologies : HTML,CSS,JavaScript,Bootstrap,Angular JS
```

Enum

The Enums are used to define a collection of constant values. One an Enum is declared with some value they are final and they cannot be modified.

Let's understand the declaration of Enums and accessing the properties of an Enum as follows,

```
enum Month {
    'January' = 'JAN',
    'February' = 'FEB',
    'March' = 'MAR'
}

let monthName:Month = Month.January;
let output:string = `Month Name : ${monthName}`;
console.log(output);
```

In Enum, we can access the properties of it using the name of Enum.

Any

In JavaScript we can assign any type of value to a variable. We can even re-assign different types of values to a variable. We can achieve this in TypeScript by using a datatype called **any**.

Once we declare a variable with **any** data type, we can assign any type of value to it.

```
let a:any = 10;
console.log(`Type of a is : ${typeof a}`); // number

a = 'Naveen';
console.log(`Type of a is : ${typeof a}`); // string

a = true;
console.log(`Type of a is : ${typeof a}`); // boolean
```

Void

We use void data type for functions which does not return any return value. Any function with non-return value, we declare it with void data type.

In JavaScript we may not specify the type of value it returns from a functions, but in TypeScript we must have to specify the return value of any function defined.

Let's understand the usage of void as follows,

```
// Function Definition
function greet(name):void{
    let greetMsg = `Good Morning ${name}`;
    console.log(greetMsg);
}

// Function Execution
greet('Naveen'); // Good Morning Naveen
```

Functions in TypeScript

A Function is a block of code that takes in some of the arguments and does some processing and returns the results back the called function or variable.

We can define functions in TypeScript same as in JavaScript way, but here in TypeScript we can specify the type of parameters takes in and also specify the type of values may returns from a function.

Let's understand the function declaration in TypeScript as follows,

```
// Function Definition
function greetMe(name:string,msg:string):string{
    return `Hello ${name} ${msg}`;
}

// Function Execution
let output:string = greetMe('Naveen','GoodMorning');
console.log(output); // Hello Naveen GoodMorning
```

In JavaScript we can call a function with same number of arguments/parameters or with less number of arguments and also with more number of arguments, but in TypeScript we can call a function with the same number of parameters specified in the function definition.

If needs to pass more or less number of arguments, we must have to declare each function with the required number of functions.

We cannot pass more or less number of parameters to a function like JavaScript. This clearly says that the TypeScript supports the concept called function overloading.

Let's understand the concept of function overloading in TypeScript is as follows,

```
// Function Overloading in TypeScript , findMax of 1 , 2 , 3 numbers
function max(a:number):number;
function max(a:number , b:number):number;
function max(a:number , b:number , c:number):number;
function max(a:number , b?:number , c?:number):number{
    if(b === undefined && c === undefined){
        return a;
    }
    else if(c === undefined){
        if(a > b){
            return a;
        }
        else{
            return b;
        }
    }
    else{
        if(a > b && a> c){
            return a;
        }
        else if(b > c){
            return b;
        }
        else{
            return c;
        }
    }
}

output = "The max of 10 is : " + max(10);

output = "The max of 10 , 20 is : " + max(10,20);

output = "The max of 10 , 20 , 30 is : " + max(10,20,30);

//Not possible
//output = "The max of 10 , 20 , 30 , 40 is : " + max(10,20,30,40);
```

As per the above example, we needs to declare the overloaded function first and then we needs to provide implementation for them.

Interfaces in TypeScript

An interface defines a contract between the specification and the actual implementation.

We can just specify the typing information inside an interface, and this can be implemented inside classes for those interfaces.

An interface defines an abstraction and that can be implemented using Classes in TypeScript. An interface just defines the rules which can be implemented in Classes of TypeScript.

Let's understand the usage of Interface in TypeScript is as follows,

We can define an interface in TypeScript is by using the keyword called 'interface'.

```
interface Student{
    name : string;
    age : number;
    group : string;
    address : {
        city : string;
        state : string;
        country : string;
    }
}
```

We have a function which accepts a student as a parameter and which is of type Student Interface.

```
function printStudent(student:Student){
    let output = JSON.stringify(student);
    console.log(output);
}
```

Here we needs to create a student object same as the structure defined in the Student Interface and pass it to this function.

```
let student1:Student = {
    name : "John",
    age : 25,
    group : "CSC",
    address : {
        city : "Hyderabad",
        state : "Telangana",
        country : "India"
    }
};
printStudent(student1);
```

We use Interface to define the rules and the structure of an Object to be passed in to any function.

Interfaces may contains methods with only declaration and no implementation will be provided inside an interface. We must have to implement an interface to a class and implement those methods.

Classes in TypeScript

In order to develop any software system in object oriented way or approach, we must needs to create few classes to represent a blueprint of various components of a software system.

A class is just a template or blue print and allow to create few objects to perform operations on the data. A class contains few properties to holds data of the system and also contains few methods to perform some operations on the data.

We normally create Objects for those blueprint classes and manipulate the data using the properties and methods of that particular class. Once any object is created for any class then all the properties and methods are available to the objects to manipulate the data.

We can also write an independent classes or first we create an interface for providing rules and we may implement those rules of an interface inside a class by implementing an interface.

Let's understand the concept of classes in TypeScript and creation of Objects to manipulate the data of a particular class.

We can create a class by using the keyword called 'class'.

```
1  class Greeter{
2
3      private greeting : string;
4
5      constructor(name : string){
6          this.greeting = name;
7      }
8
9      public greet(){
10         return "Good Morning " + this.greeting;
11     }
12
13     public get getGreetMessage(){
14         return this.greeting;
15     }
16
17     public set setGreetMessage(name : string){
18         this.greeting = name;
19     }
20 }
```

```
21
22  let greeter = new Greeter("Naveen");
23  let output = greeter.greet();
24  console.log(output);
25
26  greeter.setGreetMessage = " John";
27  output = greeter.getGreetMessage;
28  console.log(output);
```

In the Above example, line number specifies the creation of a class in TypeScript.

Line number 3, specifies creation of properties of a class.

Line number 5-7 indicates a constructor, a constructor is just like a normal function / method which is executed whenever we create an object to a class and to provide a default implementation or data to an object.

Line number 13 – 16, we call it as getter methods, which are used to access the properties of an Object. In the concept called encapsulation we should not have direct access to any variables or properties of a classes. We can only access the properties using getter methods.

Line number 17 – 19 indicates a setter methods to set values to the properties of a class.

We can even create a class by implementing an interface as follows,

```
1  interface Car{
2      description():void;
3      specification():void;
4  }
5
6  class PetrolCar implements Car{
7      private made : string;
8      private model : number;
9      private engineCapasity : string;
10     private milage : string;
11
12     constructor(made:string , model:number, engineCapasity:string,milage:string){
13         this.made = made;
14         this.model = model;
15         this.engineCapasity = engineCapasity;
16         this.milage  = milage;
17     }
```

```
18
19      description():void{
20          console.log("This is the description method of Car");
21      }
22
23      specification():void{
24          console.log("This is the specification method of Car");
25      }
26
27      get getModel():number{
28          return this.model;
29      }
30
31      set setModel(model){
32          this.model = model;
33      }
34  }
```

```
35
36  let petrolCar = new PetrolCar('Marthi Suziki',2018,'1300CC','20kmpl');
37  let out:number = petrolCar.getModel;
38  let output = "The car model is : " + petrolCar.getModel;
39  console.log(output);
40  document.getElementById('display').innerHTML = output;
```

Usage of Access Modifiers

The mainly supported modifiers in TypeScript are as follows,

1. Private
2. Protected
3. Public

1) **Private**: This is the most restricted modifier.

This we can use for properties or methods.

Any member which is declared with the private property, it will not be accessed outside of the class where it is declared.

```
1  class greet{
2      private greetMsg:string;
3      private welcomeMsg:string;
4  }
```

NOTE: For getter and setter methods, the access modifier should be same. If we try to change it, we may get compile time error.

2) **Protected**:

This is a least restricted modifier than Private.

This can be used only in the base class.

The protected modifiers can be access in the same class and its implemented/child class only.

```
class greet{
    protected greetMsg:string;
    protected welcomeMsg:string;
}
```

3) **Public**: This is the lease restricted modifier among all the other two modifiers.

Any member declared with this modifier even outside if the class also we can access them.

```
class greet{
    public greetMsg:string;
    public welcomeMsg:string;
}
```

Generics

A major part of software engineering is building components that not only have well-defined and consistent APIs, but are also reusable. Components that are capable of working on the data of today as well as the data of tomorrow will give you the most flexible capabilities for building up large software systems.

In languages like C# and Java, one of the main tools in the toolbox for creating reusable components is generics that is, being able to create a component that can work over a variety of types rather than a single one. This allows users to consume these components and use their own types.

Hello World of Generics

To start off, let's do the "hello world" of generics: the identity function. The identity function is a function that will return back whatever is passed in. You can think of this in a similar way to the echo command.

Without generics, we would either have to give the identity function a specific type:

```
function identity(arg: number): number {
    return arg;
}
```

This function takes in a number and do same calculations and returns a number back again.

In future if this same method/component has to accept any other type as well.

```
function identity(arg: any): any {
    return arg;
}
```

We can define this 'any' type but here actually we are missing the behavior. So it takes in 'any' type so that can be string/number/object. But we cannot guarantee that whatever comes in we are returning the same.

In order to avoid this, we will be using 'generics' concepts.

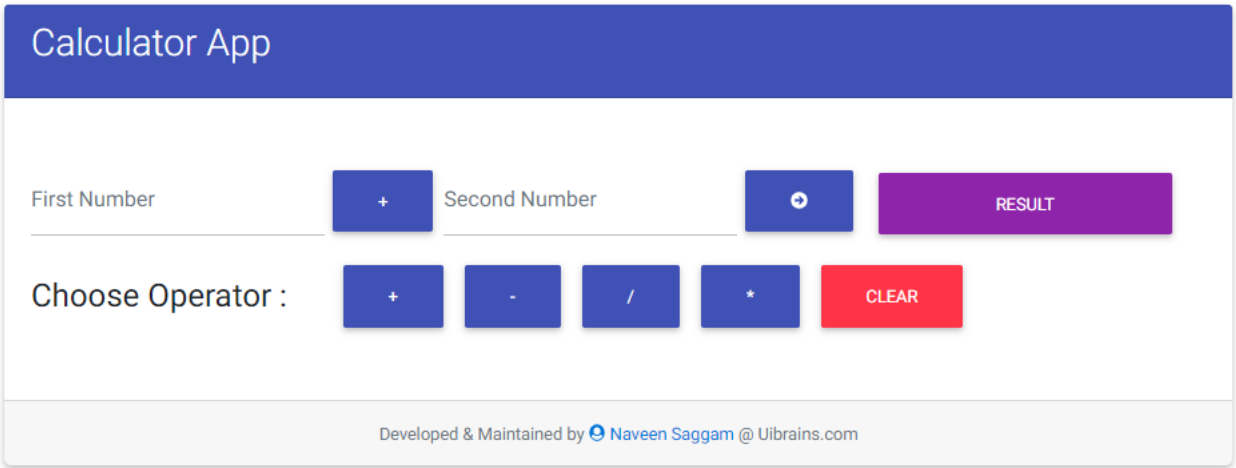
```
function identity<T>(arg: T): T {
    return arg;
}
```

In the above, function takes in a 'generic' type and do same calculations inside and returns the same type we passed in. This gives the guarantee that provided type is same as returned type.

Mini Project

In this Mini Project, we can create an app called Simple Calculator App by using all the Object Oriented concepts we learned as of now.

The App looks as follows,



Bootstrap

Bootstrap is free and open source and most popular HTML, CSS JavaScript Framework for creating responsive web applications. With the Bootstrap we can do the faster web development with responsive layouts and with mobile first approaches.

Bootstrap contains lot of CSS classes for making the web development faster, cleaner and more elegant way of developing responsive web applications. Bootstrap works with any browser and any operating system and any device.

Any developer with very less knowledge of UI Design skills can also develop an awesome websites using Bootstrap. Bootstrap is easy to learn and apply the features of it and achieve the web development faster.

Bootstrap contains HTML and CSS based design templates for typography, form , buttons , navigation and cards and other UI Components.

Introduction

Bootstrap is a free, Open Source and Worlds most popular Front End framework for creating responsive and mobile first applications.

Bootstrap features:

- ➔ Bootstrap is an Open Source
- ➔ HTML , CSS , JS framework
- ➔ Used for creating responsive applications
- ➔ Uses Mobile first approach
- ➔ Works for any browser and any Operating System
- ➔ We can develop any kind of Web Applications without UI design knowledge
- ➔ Easy to learn

Open Source:

Bootstrap is an open source means its original source code is made available as free of cost.

We can find the original source code in the following location:

https://github.com/twbs/bootstrap/tree/v4

Anyone can clone it or download it for any kind of modification and also for re distribution.

There is a redistribution of bootstrap is available as Material Design Bootstrap (MDB).

We can find the material design for bootstrap 4 is available at,

UIBrains.com

https://mdbootstrap.com/

Note:
As this is a licensed version, only some of the components are made available as free and we can download them as CSS files of MDB and start using them in our project for better UI look and feel.

HTML, CSS and JS Framework.

For any web application development the core technologies are HTML, CSS and JavaScript.

In order to develop any web applications using core technologies, requires an extensive knowledge of UI design skills and needs the best design patterns to make a web application to work more elegantly.

This also requires a lot of skeleton code base to frame any web application.

As Bootstrap is built on top of all these core technologies HTML, CSS and JavaScript and having built-in classes made available such as form , alerts , Navbars , buttons , list group , carousel and many more components with the best design patterns used in it.

By using Bootstrap, Web application development is made easy without any knowledge of UI design skills and also with the less number of lines of code.

Responsive Web Design (RWD):

This is one of the best approach of web design which makes web applications to render well on all kinds of devices such as large screen, laptops, tablets, mobiles.

This responsiveness can be achieved by dividing the whole screen with a responsive grid layout and certain media queries can be used.

```
// Extra small devices (portrait phones, less than 576px)
// No media query since this is the default in Bootstrap

// Small devices (landscape phones, 576px and up)
@media (min-width: 576px) { ... }

// Medium devices (tablets, 768px and up)
@media (min-width: 768px) { ... }

// Large devices (desktops, 992px and up)
@media (min-width: 992px) { ... }

// Extra large devices (large desktops, 1200px and up)
@media (min-width: 1200px) { ... }
```

Based on these media queries, the screens are divided into 4 sections in bootstrap,

Sm – small screen

Md – medium screen

Lg – large screen

Xl – extra large screen.

In Bootstrap all the components are built using the above media breakpoints.

Responsive Screen vs Non Responsive screen

Mobile first approach

Mobile first approach is a way of designing web sites for smaller screens and then progressively adjust the design for larger screens using CSS media queries.

Luke Wroblewski first introduced the term "Mobile first" to the Web World in 2009.

Mobile first approach is important, because in the today's world everything is mobile based.

Responsive web design vs. Mobile First Responsive web design

Responsive web design(RWD) is an approach to identify the Client device and changing the layout to fits the screen with Media Queries. In RWD, first we will design for Larger screen and Change the layout for smaller screens.

Mobile First Responsive design (Mobile First RWD) is just opposite to Responsive design. In this method, we will design for Smaller screen first, then change the layout with CSS media Queries when the screen Expands.

In Simple Words,

RWD is – Desktop -> Tablet -> Mobile

Mobile First RWD is – Mobile -> Tablet -> Desktop

Note:

The best way of designing a website suits to today's world is Mobile First RWD.

As this is an approach to reach any websites to more number of people, as 77% of the world's population has a mobile devices.

Works for any browser and any Operating System

Any application built with Bootstrap will work the same way in all the modern browsers and all the operating systems available as to today.

Note:

Without any extensive knowledge of UI design skills we can develop any kind of websites with Mobile first RWD approach.

All the bootstrap classes are self-understandable and easy to use with HTML elements and also Bootstrap is an easy to learn framework.

History of Bootstrap:

Twitter Employee Mark Otto and Jacob Thornton developed this Bootstrap and initially named as Twitter Bootstrap.

The initial release of Bootstrap is Bootstrap 1.0 in Aug, 19, 2011.
The current version of Bootstrap is Bootstrap **4.0.0** released in **January 18, 2018**.

Before Bootstrap everyone use their own code and libraries for each components. This leads to inconsistency and high maintenance burden. In order to solve this problem Mark Otto and Jacob Together developed this Bootstrap, it took them several months and finally released

the initial version in 2011. Later many other developers added to it. Later Twitter Blueprint is renamed to Bootstrap.

Bootstrap supports all most all the modern browsers like google chrome, Firefox, IE, Opera Safari etc. and it provides a set of stylesheets and enhances the default styling of HTML Elements, it is highly customizable.

There are lot of Differences between Bootstrap 3 and Bootstrap 4. In this book we will be learning Bootstrap 4 version. Bootstrap gained its popularity in the last 3 years and even all the CMS websites frameworks like WordPress, Drupal, and Joomla uses Bootstrap.

Bootstrap Summary:

Creator	Mark Otto , Jacob Thronton
Initial Release	Bootstrap 1.0 in 2011
Current Release	Bootstrap 4.0 in Jan, 2018
License	Open Source
Official website	http://getbootstrap.com
Popularity	More than 1 lakh stars in Github
Core Concepts	Mobile first and Responsive in Nature
Works well in	Any Browser , Any Device , Any Operating System

Installation of Bootstrap:

There are 3 ways we can install Bootstrap.

1. Using npm
2. Bootstrap CDN
3. Offline Version

Using NPM

This way of installing is by using node package manager.

For this we should have already installed node JS from http://nodejs.org .

Once the node js is successfully installed , go to command prompt and execute the following command for bootstrap installation.

```
C:\Users\hp pc>npm install -g bootstrap
```

Bootstrap CDN

Here in this way of installing Bootstrap we should give the CDN link to our html pages.
CDN means Content Delivery Network , where the required bootstrap required files will be hosted and once we specify the links, they can be loaded very fast from CDN network and they even stored in the browser cache for faster loading.
We should give the cdn links to the following files.
https://getbootstrap.com/docs/4.0/getting-started/download/

1. bootstrap.min.css
2. JQuery.min.js
3. Popper.js
4. Bootstrap.min.js

Offline version

In this we can manually download the mentioned files and add them to our HTML files. This method works even without internet also.
We can download from the url as below,

https://getbootstrap.com/docs/4.0/getting-started/download/

Compiled CSS and JS

Download ready-to-use compiled code for **Bootstrap v4.0.0** to easily drop into your project, which includes:

- Compiled and minified CSS bundles (see CSS files comparison)
- Compiled and minified JavaScript plugins

This doesn't include documentation, source files, or any optional JavaScript dependencies (jQuery and Popper.js).

Download

Once we click on the above download button, it downloads a zip file.
Once downloaded extract it, and got to dist folder of it to get the above files. Finally we needs to link to these files from our source code project.

Note

1. Bootstrap.min.css should be added anywhere in the head section but before </head> tag.

2. If we have our own *'styles.css'* then that should be added after bootstrap.min.css because we needs to override the existing styles of bootstrap in the styles.css.
3. Jquery.min.js, popper.min.js, bootstrap.min.js should be added in the body section just before we closing </body> tag in the HTML file.
4. While added the above JavaScript files we must follow the same order as below,
    a. Jquery.min.js
    b. Popper.min.js
    c. Bootstrap.min.js
5. Why the order is important is bootstrap is depend in popper library and which is again depend on JQuery library.
6. If we have our own *'app.js'* file to add some more JavaScript features, then that file should be added at the last after the bootstrap.min.js.

Sample Bootstrap Application (offline version)

```
1  <!doctype html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <meta name="viewport" content="width=device-width, initial-scale=1.0">
6      <meta http-equiv="X-UA-Compatible" content="ie=edge">
7      <link rel="stylesheet" href="bootstrap/css/bootstrap.min.css">
8      <link rel="stylesheet" href="bootstrap/css/style.css">
9      <title>First App</title>
10 </head>
11 <body>
12 <h1>Hello Bootstrap</h1>
13
14
15     <!-- jQuery first, then Popper.js, then Bootstrap JS -->
16     <script src="bootstrap/js/jquery-3.2.1.min.js"></script>
17     <script src="bootstrap/js/popper.min.js"></script>
18     <script src="bootstrap/js/bootstrap.min.js"></script>
19 </body>
20 </html>
```

Note

1. At line number 5, we have added a <meta> tag , which is responsible to provide responsiveness to any webpage.
2. *width="device-width"* This informs the bootstrap to capture the device width.
3. *Initial-scale = 1* this is the scaling information which is useful while was changing the device width and the scaling is initialized to 1.
4. *Shrink-to-fit = no* this attribute specifies, while we are on small screen devices it informs bootstrap weather or not to shrink with the content.

Let's discuss Bootstrap usage in detail as follows,

1. Basic Typography in Bootstrap

Bootstrap provides various typography related classes as follows,

Display classes

There are 4 display classes available to use with any html elements to display the text bigger than original.

display-1
display-2
display-3
display-4

All the display classes transforms the text to little more bigger text than original text, here display-1 is used for the bigger text and display-4 is the least big text.

```
<h1 class="display-1">My Heading</h1>
<h1 class="display-2">My Heading</h1>
```

My Heading

My Heading

Paragraphs

The *lead* class is used for little bigger text to the paragraph.

```
<p class="lead">Lorem ipsum dolor sit amet</p>
```

The *text-muted* is used for some dark shaded text.

```
<p class="text-muted">Lorem ipsum dolor sit amet</p>
```

Text Transformation

There are some classes available to transform the text to lowercase or uppercase or capitalize.

text-capitalize --> to convert from lowercase to capitalize
text-uppercase -> to convert from lowercase to uppercase

text-lowercase -> to convert from uppercase to lowercase.

```
<p class="text-lowercase">HELLO WORLD</p>
<p class="text-uppercase">hello world</p>
<p class="text-capitalize">hello world</p>
```

Style Classes

Styles classes are used for font related styling.

font-weight-bold - > to display bold Text

font-weight-normal -> to display normal text

font-italic –> to display italic text

```
<p class="font-weight-bold">Bold Text</p>
<p class="font-weight-normal">Normal Text</p>
<p class="font-italic">Italic Text</p>
```

Block Quote classes

Blockquote - > to display in quotations format (normally used for any quotations.)

```
<blockquote class="blockquote">
  <p>Lorem ipsum dolor sit amet, consectetur adipisicing elit. Nobis, rerum!</p>
</blockquote>
```

Un styled List

list-unstyled -> to removes DOTS of an un ordered list.

Inline list

list-inline -> to display a list inline

list-inline-item -> used to specify the list item in inline

```
<ul class="list-unstyled">
  <li>Lorem ipsum dolor sit amet.</li>
  <li>Lorem ipsum dolor sit amet.</li>
  <li>Lorem ipsum dolor sit amet.</li>
  <li>Lorem ipsum dolor sit amet.</li>
</ul>
```

2. Text Alignment and Display

The following Bootstrap classes are used for various text alignments and display.

To Justify the content

Text-justify -> to adjust or spread the content to all the lines of the paragraph equally.

> Lorem ipsum dolor sit amet, consectetur adipisicing elit. Dolor ad deleniti, placeat cumque, maiores fugiat tenetur ea voluptas voluptatem odio odit, dicta. Eaque reiciendis sed placeat omnis doloribus, corrupti eius.

Text alignment

Text-left - > to move align the text to left side (by default text aligned to left side).
Text-center -> used for center align the text
Text-right -> used for right alignment of text.

```
<p class="text-left">Lorem ipsum dolor sit.</p>
<p class="text-center">Lorem ipsum dolor sit.</p>
<p class="text-right">Lorem ipsum dolor sit.</p>
```

Responsive alignment

Bootstrap comes with responsive alignment classes, which specifies to align the text with respect to the device width, weather to align the text center in small scale devices.

Responsive align Right

text-sm-right -> align right in small devices (mobiles)
text-md-right -> align right in medium devices (tabs)
text-lg-right -> align right in large devices (laptops & desktops)
text-xl-right -> align right in extra-large devices (big TV's and Projectors)

```
<p class="text-sm-right">Right aligned on small or larger</p>
<p class="text-md-right">Right aligned on medium or larger</p>
<p class="text-lg-right">Right aligned on large or larger</p>
<p class="text-xl-right">Right aligned on xl or larger</p>
```

Responsive align Left

text-sm-left -> align left in small devices (mobiles)
text-md-left -> align left in medium devices (tabs)

text-lg-left -> align left in large devices (laptops & desktops)
text-xl-left -> align left in extra large devices (big TV's and Projectors)

```
<p class="text-sm-left">Left aligned text on small or larger</p>
<p class="text-md-left">Left aligned text on medium or larger</p>
<p class="text-lg-left">Left aligned text on large or wider</p>
<p class="text-xl-left">Left aligned text on extra large or wider</p>
```

Responsive Align Center

text-sm-center -> align center in small devices (mobiles)
text-md-center -> align center in medium devices (tabs)
text-lg-center -> align center in large devices (laptops & desktops)
text-xl-center -> align center in extra large devices (big TV's and Projectors)

```
<p class="text-sm-center">Center aligned text on small or larger</p>
<p class="text-md-center">Center aligned text on medium or larger</p>
<p class="text-lg-center">Center aligned text on large or wider</p>
<p class="text-xl-center">Center aligned text on extra large or wider</p>
```

NOTE

Please note the following notations used for small devices to extra-large devices.
Sm – Small Devices
Md – Medium Devices
Lg – Large Devices
Xl – Extra Large Devices

Vertical Alignment

The following Bootstrap classes are used to align the text vertically.
align-baseline -> to Align the text to the baseline
align-top -> to align the text to the top
align-bottom -> to align the text to the bottom
align-middle -> to align the text to the middle
align-text-top -> to align the text to the top
align-text-bottom -> to align the text to the bottom

```
<span class="align-baseline">baseline</span>
<span class="align-top">top</span>
<span class="align-bottom">bottom</span>
<span class="align-middle">middle</span>
<span class="align-text-top">text-top</span>
<span class="align-text-bottom">text-bottom</span>
```

Convert Block to Inline

d-inline - > Used to convert any block level element to Inline.

```
<h1 class="bg-success d-inline">Hello</h1>
<h1 class="bg-success d-inline">Goodbye</h1>
```

Hello Goodbye

Convert Inline to Block

d-block -> Used to convert any Inline element to Block level Element

```
<span class="bg-info d-block">Block</span>
```

Block

Inline-Block

d-inline-block -> to display an element in inline-block (used in main navigation)

```
<div class="bg-warning d-inline-block">
  <h3>Hello</h3>
  This is inline
</div>
<div class="bg-warning d-inline-block">
  <h3>Hello</h3>
  This is inline
</div>
```

Hello Hello

This is inline This is inline

3. Float Positions

These classes are useful for floating of a text on the browser wither right or left.

Float classes

Float-left -> to float to right
Float-right -> to float to right
Float-none -> to float none

```
<div class="float-left">Float left</div><br>
<div class="float-right">Float right</div><br>
<div class="float-none">Float none</div><br>
```

Responsive float right classes

float-sm-right -> to float right in small devices
float-md-right -> float to right in medium devices
float-lg-right -> float to right in large devices
float-xl-right -> float to right in extra-large devices

```
<div class="float-sm-right">Float right on small or wider</div><br>
<div class="float-md-right">Float right on medium or wider</div><br>
<div class="float-lg-right">Float right on large or wider</div><br>
<div class="float-xl-right">Float right on extra large or wider</div><br>
```

Responsive Left classes

float-sm-left -> to float left in small devices
float-md-left -> float to left in medium devices
float-lg-left -> float to left in large devices
float-xl-left -> float to left in extra-large devices

```
<div class="float-sm-left">Float left on small or wider</div><br>
<div class="float-md-left">Float left on medium or wider</div><br>
<div class="float-lg-left">Float left on large or wider</div><br>
<div class="float-xl-left">Float left on extra large or wider</div><br>
```

Responsive Non classes

float-sm-none -> to float none in small devices
float-md-none -> float to none in medium devices
float-lg-none -> float to none in large devices
float-xl-none -> float to none in extra-large devices

```
<div class="float-sm-none">Float none on small or wider</div><br>
<div class="float-md-none">Float none on medium or wider</div><br>
<div class="float-lg-none">Float none on large or wider</div><br>
<div class="float-xl-none">Float none on extra large or wider</div><br>
```

Clear fix

When we apply any float classes, the next elements will be placed between the floated elements. To resolve this we must use clear fix to the next elements.

Clearfix -> to clear the floating to next elements.

```
<div class="bg-success clearfix">
    <button class="btn btn-secondary float-left">Float Left</button>
    <button class="btn btn-secondary float-right">Float Right</button>
</div>
```

Fixed-top

To fix an element to the top section of the webpage. We normally use this class in main navigation.

Fixed-top -> to fix any element to top of the webpage

```
<h3 class="fixed-top">Fixed Top</h3>
```

Sticky Top

To stick an element to the top of the webpage while scrolling the page.

Sticky-top -> to stick to top

```
<h3 class="sticky-top">Sticky</h3>
```

Fixed Bottom

To fix an element to the bottom of the webpage. Normally we use this to footer section to fixed to bottom of the page.

Fixed-bottom -> to fix an element to bottom of the page.

```
<h3 class="fixed-bottom">Fixed Bottom</h3>
```

4. Colors & Backgrounds

Bootstrap comes with various classes to apply colors to the text and also apply background colors to any specific elements.

There are some primary colors specified by Bootstrap, they are as follows.

Primary -> blue color
Secondary -> gray color
Success -> green color
Info -> light blue color
Warning -> orange color
Danger -> red color
White - > white color

Text Colors

Bootstrap comes with the following list of colors to apply some colors to textual information.

Text-primary -> to display text in blue color
Text-secondary -> to display text in gray color
Text-success -> to display text in green color
Text-info -> to display text in light blue color
Text-warning -> to display text in orange color
Text-danger -> to display text in red color
Text-white -> to display text in white color

```
<p class="text-info">Lorem ipsum dolor sit amet, consectetur.</p>
<p class="text-primary">Lorem ipsum dolor sit amet, consectetur.</p>
<p class="text-success">Lorem ipsum dolor sit amet, consectetur.</p>
<p class="text-warning">Lorem ipsum dolor sit amet, consectetur.</p>
<p class="text-danger">Lorem ipsum dolor sit amet, consectetur.</p>
<p class="text-secondary">Lorem ipsum dolor sit amet, consectetur.</p>
<p class="text-white bg-success">Lorem ipsum dolor sit amet, consectetur.</p>
```

Background color

Bg-primary -> to apply blue color background to any element
Bg-secondary -> to apply gray color background to any element
Bg-success -> to apply green color background to any element
Bg-info -> to apply light blue color background to any element
Bg-warning -> to apply orange color background to any element
Bg-danger -> to apply red color background to any element
Bg-white -> to apply white color background to any element

```
<div class="bg-info">Nullam id dolor id nibh ultricies</div>
<div class="bg-primary">Duis mollis, est non commodo luctus</div>
<div class="bg-success">Maecenas sed diam eget risus varius </div>
<div class="bg-warning">Etiam porta sem malesuada </div>
<div class="bg-danger">Donec ullamcorper nulla non </div>
<div class="bg-secondary">Cras mattis consectetur</div>
<div class="bg-white">Cras mattis consectetur purus</div>
```

Text Invisible

To hide any text we use the class name invisible.
Invisible -> to make the specified text invisible

```
<h1 class="invisible">Hello</h1>
```

5. Spacing in Bootstrap

Bootstrap comes with various class names to apply spacing between elements (margin) and spacing around the content (padding).
Margin classes are represented with a letter 'M'
Padding classes are represented with a letter 'P'

Margin related classes

Each margin class is of 5 categories, they are (m-0, m-1, m-2, m-3, m-4, m-5)

M -> to apply margin all the sides
Mb -> to apply margin only to bottom
Mt -> to apply margin only to top
Ml -> to apply margin only left
Mr -> to apply margin only to right
Mx -> to apply margin to left as well as right
My -> to apply margin to top as well as bottom

```
<h1 class="bg-success mb-0">Lorem ipsum.</h1>
<h1 class="bg-success mb-1">Lorem ipsum.</h1>
<h1 class="bg-success mb-2">Lorem ipsum.</h1>
<h1 class="bg-success mb-3">Lorem ipsum.</h1>
<h1 class="bg-success mb-4">Lorem ipsum.</h1>
<h1 class="bg-success mb-5">Lorem ipsum.</h1>
```

Lorem ipsum.
Lorem ipsum.
Lorem ipsum.
Lorem ipsum.
Lorem ipsum.
Lorem ipsum.

Note: each of the above margin classes are again of 6 categories as from (mb-0 to mb-5)

Padding related classes

Each padding class is of 5 categories, they are (p-0, p-1, p-2, p-3, p-4, p-5)

p -> to apply padding all the sides
pb -> to apply padding only to bottom
pt -> to apply padding only to top
pl -> to apply padding only left

pr -> to apply padding only to right
px -> to apply padding to left as well as right
py -> to apply padding to top as well as bottom

```
<h1 class="bg-warning pb-0">Lorem ipsum.</h1>
<h1 class="bg-warning pb-1">Lorem ipsum.</h1>
<h1 class="bg-warning pb-2">Lorem ipsum.</h1>
<h1 class="bg-warning pb-3">Lorem ipsum.</h1>
<h1 class="bg-warning pb-4">Lorem ipsum.</h1>
<h1 class="bg-warning pb-5">Lorem ipsum.</h1>
```

Lorem ipsum.
Lorem ipsum.
Lorem ipsum.
Lorem ipsum.
Lorem ipsum.
Lorem ipsum.

Note: each of the above padding classes are again of 6 categories as from (pb-0 to pb-5)

6. Sizing in Bootstrap

Bootstrap comes with several width and height classes as follows, normally we use these classes to apply width and height classes for displaying images and other elements.

Width classes

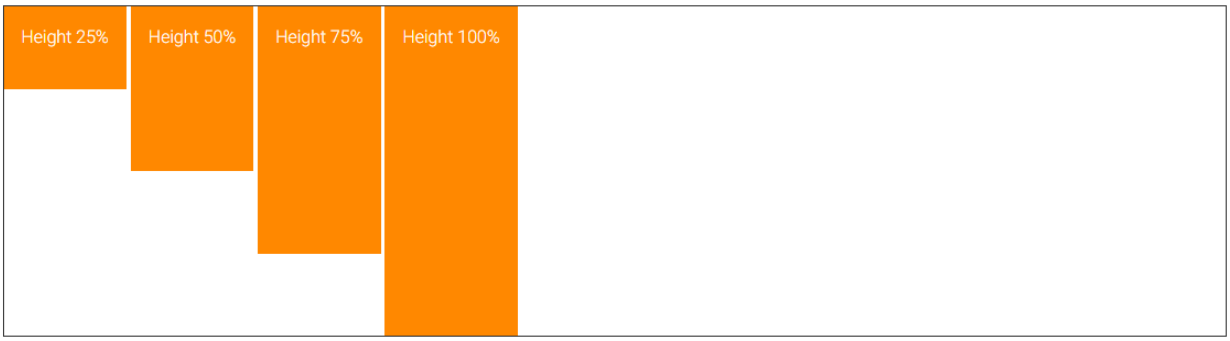
w-25 -> 25% percent of actual width
w-50 -> 50% percent of actual width
w-75 -> 75% percent of actual width
w-100 -> 100% percent of actual width

```
<div class="bg-success p-3 m-1 w-25 text-white">Width 25%</div>
<div class="bg-success p-3 m-1 w-50 text-white">Width 50%</div>
<div class="bg-success p-3 m-1 w-75 text-white">Width 75%</div>
<div class="bg-success p-3 m-1 w-100 text-white">Width 100%</div>
```

Height classes

h-25 -> 25% percent of actual Height
h-50 -> 50% percent of actual Height
h-75 -> 75% percent of actual Height
h-100 -> 100% percent of actual Height

```
<div style="height:300px;border:1px solid #333;">
    <div class="bg-warning d-inline-block p-3 h-25 text-white">Height 25%</div>
    <div class="bg-warning d-inline-block p-3 h-50 text-white">Height 50%</div>
    <div class="bg-warning d-inline-block p-3 h-75 text-white">Height 75%</div>
    <div class="bg-warning d-inline-block p-3 h-100 text-white">Height 100%</div>
</div>
```



Borders

These Borders classes can be applied to text , images , or any other elements.
In order to apply any border, we must use the base class “border”
And the sub classes are as follows,

```
<span class="border border-primary">Lorem ipsum.</span>
<span class="border border-secondary">Lorem ipsum.</span>
<span class="border border-success">Lorem ipsum.</span>
<span class="border border-danger">Lorem ipsum.</span>
<span class="border border-warning">Lorem ipsum.</span>
<span class="border border-info">Lorem ipsum.</span>
<span class="border border-light">Lorem ipsum.</span>
<span class="border border-dark">Lorem ipsum.</span>
<span class="border border-white">Lorem ipsum.</span>
```

We can even apply some border radius to specified sides as well.

```
<img src="img/image1.jpeg" class=" w-25 rounded">
<img src="img/image1.jpeg" class="w-25 rounded-top">
<img src="img/image1.jpeg" class="w-25 rounded-right">
<img src="img/image1.jpeg" class="w-25 rounded-bottom">
<img src="img/image1.jpeg" class="w-25 rounded-left">
<img src="img/image1.jpeg" class="w-25 rounded-circle">
<img src="img/image1.jpeg" class="w-25 rounded-0">
```

7. Breakpoints

Since Bootstrap is developed to be mobile first, we use a handful of media queries to create sensible breakpoints for our layouts and interfaces. These breakpoints are mostly based on minimum viewport widths and allow us to scale up elements as the viewport changes. Bootstrap primarily uses the following media query ranges—or breakpoints—in our source Sass files for our layout, grid system, and components.

// Extra small devices (portrait phones, less than 576px)

// No media query since this is the default in Bootstrap

```
// Small devices (landscape phones, 576px and up)
@media (min-width: 576px) { ... }

// Medium devices (tablets, 768px and up)
@media (min-width: 768px) { ... }

// Large devices (desktops, 992px and up)
@media (min-width: 992px) { ... }

// Extra large devices (large desktops, 1200px and up)
@media (min-width: 1200px) { ... }
```

Device	Browser width
Small devices (mobiles)	576px
Medium devices (tablets)	768px
Large devices (laptops , Desktops)	992px
Extra Large Devices (large desktops ,tv's)	1200px

8. Buttons in Bootstrap

Bootstrap includes several predefined button styles, each serving its own semantic purpose, with a few extras thrown in for more control.

```
<button type="button" class="btn btn-primary">Primary</button>
<button type="button" class="btn btn-secondary">Secondary</button>
<button type="button" class="btn btn-success">Success</button>
<button type="button" class="btn btn-info">Info</button>
<button type="button" class="btn btn-warning">Warning</button>
<button type="button" class="btn btn-danger">Danger</button>
<button type="button" class="btn btn-dark">Dark</button>
```

Button Tags

The .btn classes are designed to be used with the <button> element. However, you can also use these classes on <a> or <input> elements (though some browsers may apply a slightly different rendering).

```
<a href="#" role="button" class="btn btn-success">Link</a>
<button type="submit" class="btn btn-success">Button</button>
<input type="button" class="btn btn-success" value="Input">
<input type="submit" class="btn btn-success" value="Submit">
<input type="reset" class="btn btn-success" value="Reset">
```

LINK BUTTON INPUT SUBMIT RESET

Outline Buttons

In need of a button, but not the hefty background colors they bring? Replace the default modifier classes with the .btn-outline-* ones to remove all background images and colors on any button.

```
<button type="button" class="btn btn-outline-primary">Primary</button>
<button type="button" class="btn btn-outline-secondary">Secondary</button>
<button type="button" class="btn btn-outline-success">Success</button>
<button type="button" class="btn btn-outline-info">Info</button>
<button type="button" class="btn btn-outline-warning">Warning</button>
<button type="button" class="btn btn-outline-danger">Danger</button>
```

PRIMARY SECONDARY SUCCESS INFO WARNING DANGER

Button Sizes

Fancy larger or smaller buttons? Add .btn-lg or .btn-sm for additional sizes.

```
<button type="button" class="btn btn-success btn-lg">Large button</button>
<button type="button" class="btn btn-success btn-sm">Small button</button>
<br><br>
<button type="button" class="btn btn-secondary btn-lg btn-block">Block level
button</button>
```

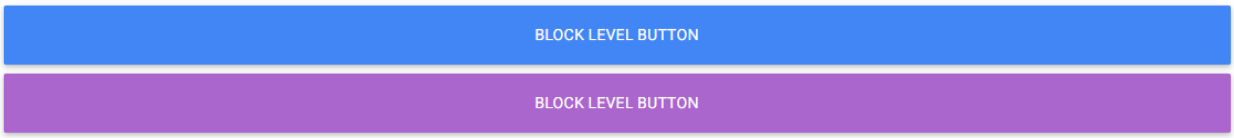
LARGE BUTTON SMALL BUTTON

BLOCK LEVEL BUTTON

Block Level buttons

Create block level buttons—those that span the full width of a parent—by adding .btn-block.

```
<button type="button" class="btn btn-primary btn-lg btn-block">Block level
button</button>
<button type="button" class="btn btn-secondary btn-lg btn-block">Block level
button</button>
```



Button States

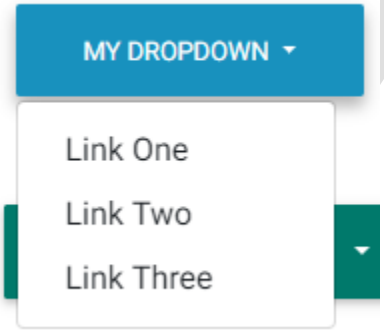
Each button contains mainly three states as follows,

```
<button type="button" class="btn btn-warning">Regular Button</button>
<button type="button" class="btn btn-warning active">Active Button</button>
<button type="button" class="btn btn-warning disabled">Disabled Button</button>
```

Button Drop Downs

Buttons are also can be used to display dropdowns as below.

```
<div class="dropdown">
    <button type="button" class="btn btn-info dropdown-toggle"
            data-toggle="dropdown">
        My Dropdown
    </button>
    <div class="dropdown-menu">
        <a href="#" class="dropdown-item">Link One</a>
        <a href="#" class="dropdown-item">Link Two</a>
        <a href="#" class="dropdown-item">Link Three</a>
    </div>
</div>
```



Split Dropdowns

```
<div class="btn-group">
    <button type="button" class="btn btn-teal">My Button</button>
    <button type="button" class="btn btn-teal dropdown-toggle"
            data-toggle="dropdown">
    </button>
    <div class="dropdown-menu">
        <a href="#" class="dropdown-item">Link One</a>
        <a href="#" class="dropdown-item">Link Two</a>
        <a href="#" class="dropdown-item">Link Three</a>
        <div class="dropdown-divider"></div>
        <a href="#" class="dropdown-item">Link Four</a>
    </div>
</div>
```

Button Groups

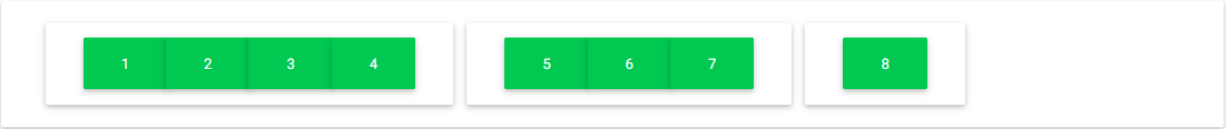
We can display number of buttons in a group as follows,

```
<div class="btn btn-group">
    <button type="button" class="btn btn-secondary">Left</button>
    <button type="button" class="btn btn-secondary">Middle</button>
    <button type="button" class="btn btn-secondary">Right</button>
</div>
```

Button Toolbar

A group of button groups is denoted as button toolbar.

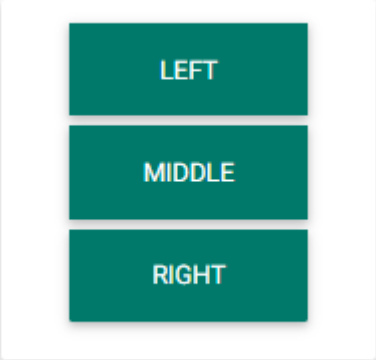
```
<div class="btn btn-toolbar">
    <div class="btn btn-group">
        <button type="button" class="btn btn-success">1</button>
        <button type="button" class="btn btn-success">2</button>
        <button type="button" class="btn btn-success">3</button>
        <button type="button" class="btn btn-success">4</button>
    </div>
    <div class="btn btn-group">
        <button type="button" class="btn btn-success">5</button>
        <button type="button" class="btn btn-success">6</button>
        <button type="button" class="btn btn-success">7</button>
    </div>
    <div class="btn btn-group">
        <button type="button" class="btn btn-success">8</button>
    </div>
</div>
```



Button Group Vertical

We can even display a vertical button group as follows,

```
<div class="btn btn-group-vertical">
    <button type="button" style="display: none" class="btn btn-teal">Left</button>
    <button type="button" class="btn btn-teal">Left</button>
    <button type="button" class="btn btn-teal">Middle</button>
    <button type="button" class="btn btn-teal">Right</button>
</div>
```



9. Navbars

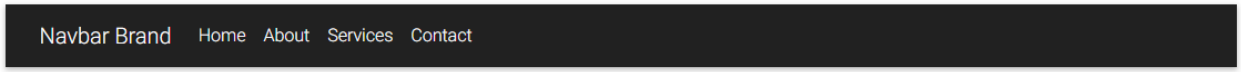
Navbars are used to display the main navigation for any websites.
Basically each Navbar comes with the following classes,

.navbar -> is the base class for navigation
.navbar-expand{-sm|-md|-lg|-xl} -> for Responsive Navbar collapse
.navbar-dark -> for dark Navbar
.navbar-brand for your company, product, or project name.
.navbar-nav for a full-height and lightweight navigation (including support for dropdowns).
.nav-item -> for each navigation item like (home , news , contact etc)
.nav-link -> each nav item contains a nav link
.navbar-toggler for use with our collapse plugin and other navigation togglingbehaviors.
.form-inline for any form controls and actions.
.navbar-text for adding vertically centered strings of text.
.collapse.navbar-collapse for grouping and hiding navbar contents by a parent breakpoint.

Simple Navbar Example

```
<nav class="navbar navbar-expand-sm navbar-dark bg-dark">
    <div class="container">
        <a href="#" class="navbar-brand">
            Navbar Brand </a>
        <div class="collapse navbar-collapse" id="simpleNavbar">
            <ul class="navbar-nav">
                <li class="nav-item">
                    <a class="nav-link"  href="#">Home</a>
                </li>
                <li class="nav-item">
                    <a class="nav-link" href="#">About</a>
                </li>
                <li class="nav-item">
                    <a class="nav-link" href="#">Services</a>
                </li>
                <li class="nav-item">
                    <a class="nav-link"  href="#">Contact</a>
                </li>
            </ul>
        </div>
    </div>
</nav>
```

SIMPLE NAVBAR



Responsive Navbar

```
<nav class="navbar navbar-expand-sm navbar-dark bg-dark">
    <div class="container">
        <a href="#" class="navbar-brand">
            Navbar Brand </a>
        <button class="navbar-toggler" data-toggle="collapse" data-target="#responsiveNavbar">
            <span class="navbar-toggler-icon"></span>
        </button>
         <div class="collapse navbar-collapse" id="responsiveNavbar">
             <ul class="navbar-nav">
                 <li class="nav-item">
                     <a class="nav-link"  href="#">Home</a>
                 </li>
                 <li class="nav-item">
                     <a class="nav-link" href="#">About</a>
                 </li>
                 <li class="nav-item">
                     <a class="nav-link" href="#">Services</a>
                 </li>
                 <li class="nav-item">
                     <a class="nav-link"  href="#">Contact</a>
                 </li>
             </ul>
         </div>
    </div>
</nav>
```

NAVBAR WITH RESPONSIVE TOGGLE

Navbar with Form inline

```
<nav class="navbar navbar-expand-lg navbar-dark bg-dark">
    <div class="container">
        <a href="#" class="navbar-brand">
            Navbar Brand </a>
        <button class="navbar-toggler" data-toggle="collapse" data-target="#formNavbar">
            <span class="navbar-toggler-icon"></span>
        </button>

        <div class="collapse navbar-collapse" id="formNavbar">
            <ul class="navbar-nav">
                <li class="nav-item">
                    <a class="nav-link"  href="#">Home</a>
                </li>
                <li class="nav-item">
                    <a class="nav-link" href="#">About</a>
                </li>
                <li class="nav-item">
                    <a class="nav-link" href="#">Services</a>
                </li>
                <li class="nav-item">
                    <a class="nav-link"  href="#">Contact</a>
                </li>
            </ul>
            <form class="form-inline ml-auto">
                <input class="form-control" type="text" placeholder="Search...">
                <button type="submit" class="btn btn-outline-success btn-sm">Search</button>
            </form>
        </div>
    </div>
</nav>
```

NAVBAR WITH FORM

Navbar with Drop Down

```
<nav class="navbar navbar-expand-lg navbar-dark bg-dark">
    <div class="container">
        <a href="#" class="navbar-brand">
            Navbar Brand </a>
        <button class="navbar-toggler" data-toggle="collapse" data-target="#dropdownNavbar">
            <span class="navbar-toggler-icon"></span>
        </button>

        <div class="collapse navbar-collapse" id="dropdownNavbar">
            <ul class="navbar-nav">
                <li class="nav-item">
                    <a class="nav-link"  href="#">Home</a>
                </li>
                <li class="nav-item">
                    <a class="nav-link" href="#">About</a>
                </li>
                <li class="nav-item">
                    <a class="nav-link" href="#">Services</a>
                </li>

                <li class="nav-item">
                    <a class="nav-link"  href="#">Contact</a>
                </li>
                <li class="nav-item dropdown">
                    <a class="nav-link dropdown-toggle"
                       data-toggle="dropdown" href="#">Courses</a>
                    <div class="dropdown-menu">
                        <a class="dropdown-item" href="#">HTML</a>
                        <a class="dropdown-item" href="#">JavaScript</a>
                        <a class="dropdown-item" href="#">CSS</a>
                        <a class="dropdown-item" href="#">Bootstrap</a>
                    </div>
                </li>
            </ul>
            <form class="form-inline ml-auto">
                <input class="form-control" type="text" placeholder="Search...">
                <button type="submit" class="btn btn-outline-success btn-sm">Search</button>
            </form>
        </div>
    </div>
</nav>
```

NAVBAR WITH DROPDOWN

10. List Groups

List groups are a flexible and powerful component for displaying a series of content. Modify and extend them to support just about any content within.

The most basic list group is an unordered list with list items and the proper classes. Build upon it with the options that follow, or with your own CSS as needed.

```
<ul class="list-group">
    <li class="list-group-item list-group-item-action">My List Item One</li>
    <li class="list-group-item list-group-item-action">My List Item Two</li>
    <li class="list-group-item list-group-item-action">My List Item Three</li>
    <li class="list-group-item list-group-item-action">My List Item Four</li>
    <li class="list-group-item list-group-item-action">My List Item Five</li>
</ul>
```

My List Item One

My List Item Two

My List Item Three

My List Item Four

My List Item Five

Active Items

Add .active to a .list-group-item to indicate the current active selection.

```
<ul class="list-group">
     <a class="list-group-item list-group-item-action" href="#">My List Item One</a>
     <a class="list-group-item list-group-item-action" href="#">My List Item Two</a>
     <a class="list-group-item list-group-item-action" href="#">My List Item Three</a>
     <a class="list-group-item list-group-item-action" href="#">My List Item Four</a>
     <a class="list-group-item list-group-item-action" href="#">My List Item Five</a>
 </ul>
```

Disabled Items

Add .disabled to a .list-group-item to make it *appear* disabled. Note that some elements with .disabled will also require custom JavaScript to fully disable their click events (e.g., links).

```
<ul class="list-group">
  <li class="list-group-item disabled">Cras justo odio</li>
  <li class="list-group-item">Dapibus ac facilisis in</li>
  <li class="list-group-item">Morbi leo risus</li>
  <li class="list-group-item">Porta ac consectetur ac</li>
  <li class="list-group-item">Vestibulum at eros</li>
</ul>
```

Links and Buttons

Use <a>s or <button>s to create *actionable* list group items with hover, disabled, and active states by adding .list-group-item-action. We separate these pseudo-classes to ensure list groups made of non-interactive elements (like <li>s or s) don't provide a click or tap affordance.

```
<div class="list-group">
  <a href="#" class="list-group-item list-group-item-action active">
    Cras justo odio
  </a>
  <a href="#" class="list-group-item list-group-item-action">Dapibus ac facilisis in</a>
  <a href="#" class="list-group-item list-group-item-action">Morbi leo risus</a>
  <a href="#" class="list-group-item list-group-item-action">Porta ac consectetur ac</a>
  <a href="#" class="list-group-item list-group-item-action disabled">Vestibulum at eros</a>
</div>
```

Contextual Classes

Use contextual classes to style list items with a background and color.

```
<ul class="list-group">
    <li class="list-group-item bg-info text-white ">My List Item One</li>
    <li class="list-group-item bg-primary text-white">My List Item Two</li>
    <li class="list-group-item bg-success text-white">My List Item Three</li>
    <li class="list-group-item bg-warning text-white">My List Item Four</li>
    <li class="list-group-item bg-danger text-white">My List Item Five</li>
</ul>
```

With Badges

Add badges to any list group item to show unread counts, activity, and more with the help of some utilities.

```html
<ul class="list-group">
    <li class="list-group-item">My List Item One</li>
    <li class="list-group-item">My List Item Two
        <span class="badge badge-danger badge-pill">23</span></li>
    <li class="list-group-item">My List Item Three
        <span class="badge badge-success badge-pill">43</span></li>
    <li class="list-group-item">My List Item Four</li>
    <li class="list-group-item">My List Item Five</li>
</ul>
```

My List Item One

My List Item Two 23

My List Item Three 43

My List Item Four

My List Item Five

11. Forms

Bootstrap comes with various classes for each form elements.
.form-group -> each form element should be in this class.
.form-control -> each form element is represented with this class.

```html
<form>
    <div class="form-group">
        <label for="name">Name</label>
        <input class="form-control" type="text" id="name" placeholder="Enter name">
    </div>
    <div class="form-group">
        <label for="email">Email address</label>
        <input class="form-control" type="email" id="email" placeholder="Enter email">
        <small class="text-muted">Your email will not be shared with anyone</small>
    </div>
    <div class="form-group">
        <label for="password">Password</label>
        <input class="form-control" type="password" id="password" placeholder="Password" readonly>
    </div>
    <div class="form-group">
        <label for="gender">Gender</label>
        <select class="form-control" id="gender">
            <option>Male</option>
            <option>Female</option>
        </select>
    </div>
```

```
    <div class="form-group">
        <label for="message">Message</label>
        <textarea class="form-control" id="message" rows="3"></textarea>
    </div>
    <div class="form-group">
        <label for="file">File input</label>
        <input class="form-control-file" type="file" id="file">
        <small id="fileHelp" class="text-muted">Max 3mb size</small>
    </div>

    <fieldset>
        <legend>What are you using this service for?</legend>
        <div class="form-check">
            <label class="form-check-label">
                <input class="form-check-input" type="radio" value="option1" checked>
                Personal
            </label>
        </div>
        <div class="form-check">
            <label class="form-check-label">
                <input class="form-check-input" type="radio" value="option2">
                Commercial Use
            </label>
        </div>
        <div class="form-check">
            <label class="form-check-label">
                <input class="form-check-input" type="radio" value="option3" disabled>
               Testing
            </label>
        </div>
    </fieldset>
    <div class="form-check">
        <label class="form-check-label">
            <input class="form-check-input" type="checkbox">
                Sign up for newsletter
        </label>
    </div>
    <button class="btn btn-success" type="submit">Submit</button>
</form>
```

Name

Enter name

Email address

Enter email

Your email will not be shared with anyone

Password

Password

Gender

Male

Message

File input

Choose File No file chosen

Max 3mb size

What are you using this service for?

- Personal
- Commercial Use
- Testing
- Sign up for newsletter

SUBMIT

Inline forms

Forms can be displayed inline as follows,

```
<form class="form-inline">
    <input class="form-control mr-3" type="text" id="username" placeholder="Enter username">
    <input class="form-control mr-3" type="text" id="password" placeholder="Password">
    <div class="form-check mr-3">
        <label class="form-check-label">
            <input class="form-check-input" type="checkbox"> Remember me
        </label>
    </div>

    <button class="btn btn-success" type="submit">Submit</button>
</form>
```

Enter username Password Remember me SUBMIT

12. Input Groups

Input groups are used to extend the form controls by adding text , buttons , button groups on either side of the textual inputs.

Place one add-on or button on either side of an input. You may also place one on both sides of an input.

```
<div class="input-group mb-3">
    <div class="input-group-prepend">
        <span  id="basic-addon1" class="input-group-text bg-danger text-white">@</span>
    </div>
    <input class="form-control bg-dark text-white" type="text" placeholder="Username" aria-
label="Username" aria-describedby="basic-addon1">
</div>

<div class="input-group mb-3">
    <input class="form-control bg-dark text-white" type="text" placeholder="Recipient's username"
aria-label="Recipient's username" aria-describedby="basic-addon2">
    <div class="input-group-append">
        <span class="input-group-text bg-danger text-white"  id="basic-addon2">@example.com</span>
    </div>
</div>
```

@ Username

Recipient's username @example.com

```
<label for="basic-url">Your vanity URL</label>
<div class="input-group">
    <div class="input-group-prepend">
        <span class="input-group-text bg-danger text-white"  id="basic-addon3">https://example.com
/users/</span>
    </div>
    <input class="form-control bg-dark text-white" type="text" id="basic-url" aria-describedby="basic-
addon3">
</div>
```

Your vanity URL

https://example.com/users/

Input groups for checkboxes and Radio buttons

We can add input groups for checkboxes and radio buttons as follows,

```
<div class="input-group mt-3">
    <div class="input-group-prepend">
        <span class="input-group-text bg-danger text-white">$</span>
    </div>
    <input class="form-control bg-dark text-white" type="text" aria-label="Amount (to the nearest
dollar)">
    <div class="input-group-append">
        <span class="input-group-text bg-danger text-white">.00</span>
    </div>
</div>
```

$ .00

```
<div class="input-group">
    <div class="input-group-prepend">
        <div class="input-group-text bg-danger text-white">
            <input type="checkbox">
        </div>
    </div>
    <input class="form-control bg-dark text-white" type="text" aria-label="Text input with checkbox">
</div>
```

```
<div class="input-group mt-3">
    <div class="input-group-prepend">
        <div class="input-group-text bg-danger text-white">
            <input type="radio">
        </div>
    </div>
    <input class="form-control bg-dark text-white" type="text" aria-label="Text input with checkbox">
</div>
```

13. Alerts

Alerts messages are used to provide contextual feedback messages for typical user actions with the handful of available and flexible alert messages.
Alerts are available for any length of text, as well as an optional dismiss button. For proper styling, use one of the eight **required** contextual classes (e.g., .alert-success).

```
<div class="alert alert-success">
     <strong>Success</strong> Blog post added <a href="#" class="alert-link">Check it out</a>
 </div>
 <div class="alert alert-primary">
     <strong>Attention</strong> You have a new message <a href="#" class="alert-link">Check it out</a>
 </div>
```

Success Blog post added **Check it out**

Attention You have a new message **Check it out**

```
<div class="alert alert-warning">
    <strong>Warning</strong> Please check the log files <a href="#" class="alert-link">Check it out</a>
</div>
<div class="alert alert-danger">
    <h4>There's a Problem</h4>
    <strong>Error</strong> Database update failed <a href="#" class="alert-link">Check it out</a>
</div>
```

Warning Please check the log files **Check it out**

There's a Problem

Error Database update failed **Check it out**

Link Color

To display the link color as per the alert messages we use the class called '.alert-link'. Use the .alert-link utility class to quickly provide matching colored links within any alert.

```
<div class="alert alert-primary" role="alert">
    This is a primary alert with <a href="#" class="alert-link">an example link</a>. Give it a click
if you like.
</div>
<div class="alert alert-secondary" role="alert">
    This is a secondary alert with <a href="#" class="alert-link">an example link</a>. Give it a click
if you like.
</div>
<div class="alert alert-success" role="alert">
    This is a success alert with <a href="#" class="alert-link">an example link</a>. Give it a click
if you like.
</div>
<div class="alert alert-danger" role="alert">
    This is a danger alert with <a href="#" class="alert-link">an example link</a>. Give it a click if
you like.
</div>
```

This is a primary alert with **an example link**. Give it a click if you like.

This is a secondary alert with **an example link**. Give it a click if you like.

This is a success alert with **an example link**. Give it a click if you like.

This is a danger alert with **an example link**. Give it a click if you like.

Dismissible Alert messages

We can make a dismissible alert messages by adding .alert-dismissible class.

```
<!-- DISMISSABLE -->
<div class="alert alert-dismissible alert-success">
    <button type="button" class="close">
        <i class="fa fa-window-close" data-dismiss="alert"></i>
    </button>
    <strong>Success</strong> Blog post added <a href="#">Check it out</a>
</div>
```

Success Blog post added Check it out

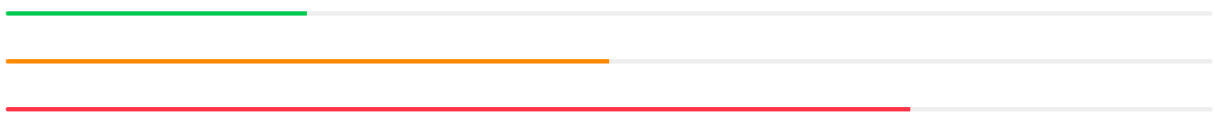
Progress Bars

Progress bars are used to display a progressing information.

Progress components are built with two HTML elements, some CSS to set the width, and a few attributes. We don't use the HTML5 <progress> element, ensuring you can stack progress bars, animate them, and place text labels over them.

- We use the .progress as a wrapper to indicate the max value of the progress bar.
- We use the inner .progress-bar to indicate the progress so far.
- The .progress-bar requires an inline style, utility class, or custom CSS to set their width.
- The .progress-bar also requires some role and aria attributes to make it accessible.

```
<div class="progress">
    <div class="progress-bar bg-success" style="width: 25%;height: 20px;">25%</div>
</div>
<br>
<div class="progress">
    <div class="progress-bar bg-warning" style="width: 50%;height: 20px;">25%</div>
</div>
<br>
<div class="progress">
    <div class="progress-bar bg-danger" style="width: 75%;height: 20px;">25%</div>
</div>
```



14. Tables in Bootstrap

Tables can be displayed with the default styling by adding the class name '.table'.

Using the most basic table markup, here's how .table-based tables look in Bootstrap. **All table styles are inherited in Bootstrap 4**, meaning any nested tables will be styled in the same manner as the parent.

```html
<table class="table">
    <thead>
        <tr>
            <th>#</th>
            <th>First Name</th>
            <th>Last Name</th>
            <th>Email</th>
        </tr>
    </thead>
    <tbody>
        <tr>
            <th scope="row">1</th>
            <td>John</td>
            <td>Doe</td>
            <td>jdoe@gmail.com</td>
        </tr>
    </tbody>
</table>
```

#	First Name	Last Name	Email
1	John	Doe	jdoe@gmail.com
2	Will	Johnson	will@yahoo.com
3	Shannon	Williams	shannon@yahoo.com

We use the following classes for table
.table -> base class to display table
.table-dark -> to display Inverse table
.table-bordered - > to display a bordered table
.table-hover -> to display Hovarable table
.table-responsive-sm -> for responsive table

```html
<table class="table table-dark">
    <thead>
        <tr>
            <th>#</th>
            <th>First Name</th>
            <th>Last Name</th>
            <th>Email</th>
        </tr>
    </thead>
    <tbody>
        <tr>
            <th scope="row">1</th>
            <td>John</td>
            <td>Doe</td>
            <td>jdoe@gmail.com</td>
        </tr>
</table>
```

#	First Name	Last Name	Email
1	John	Doe	jdoe@gmail.com
2	Will	Johnson	will@yahoo.com
3	Shannon	Williams	shannon@yahoo.com

```html
<table class="table">
    <thead class="table-dark">
        <tr>
            <th>#</th>
            <th>First Name</th>
            <th>Last Name</th>
            <th>Email</th>
        </tr>
    </thead>
    <tbody>
        <tr>
            <th scope="row">1</th>
            <td>John</td>
            <td>Doe</td>
            <td>jdoe@gmail.com</td>
        </tr>
    </tbody>
</table>
```

#	First Name	Last Name	Email
1	John	Doe	jdoe@gmail.com
2	Will	Johnson	will@yahoo.com
3	Shannon	Williams	shannon@yahoo.com

```html
<table class="table table-bordered">
    <thead class="table-dark">
        <tr>
            <th>#</th>
            <th>First Name</th>
            <th>Last Name</th>
            <th>Email</th>
        </tr>
    </thead>
    <tbody>
        <tr>
            <th scope="row">1</th>
            <td>John</td>
            <td>Doe</td>
            <td>jdoe@gmail.com</td>
        </tr>
    </tbody>
</table>
```

#	First Name	Last Name	Email
1	John	Doe	jdoe@gmail.com
2	Will	Johnson	will@yahoo.com
3	Shannon	Williams	shannon@yahoo.com

```html
<table class="table table-hover text-center">
    <thead class="bg-success text-white ">
        <tr>
            <th>#</th>
            <th>First Name</th>
            <th>Last Name</th>
            <th>Email</th>
        </tr>
    </thead>
    <tbody>
        <tr>
            <th scope="row">1</th>
            <td>John</td>
            <td>Doe</td>
            <td>jdoe@gmail.com</td>
        </tr>
    </tbody>
</table>
```

#	First Name	Last Name	Email
1	John	Doe	jdoe@gmail.com
2	Will	Johnson	will@yahoo.com
3	Shannon	Williams	shannon@yahoo.com

Pagination

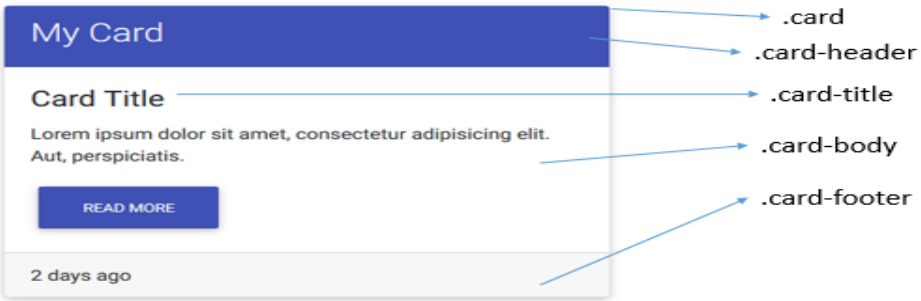
We use a large block of connected links for our pagination, making links hard to miss and easily scalable—all while providing large hit areas. Pagination is built with list HTML elements so screen readers can announce the number of available links. Use a wrapping <nav> element to identify it as a navigation section to screen readers and other assistive technologies.

In addition, as pages likely have more than one such navigation section, it's advisable to provide a descriptive aria-label for the <nav> to reflect its purpose. For example, if the pagination component is used to navigate between a set of search results, an appropriate label could be aria-label="Search results pages".

```
<nav>
    <ul class="pagination pagination-lg justify-content-start">
        <li class="page-item disabled"><a class="page-link " href="#">Previous</a></li>
        <li class="page-item active"><a class="page-link" href="#">1</a></li>
        <li class="page-item"><a class="page-link" href="#">2</a></li>
        <li class="page-item"><a class="page-link" href="#">3</a></li>
        <li class="page-item"><a class="page-link" href="#">Next</a></li>
    </ul>
</nav>
```

15. Cards

A **card** is a flexible and extensible content container. It includes options for headers and footers, a wide variety of content, contextual background colors, and powerful display options. If you're familiar with Bootstrap 3, cards replace our old panels, wells, and thumbnails. Similar functionality to those components is available as modifier classes for cards.




```
<!-- CARD BODY -->
<div class="card">
    <div class="card-body">
        Lorem ipsum dolor sit amet, consectetur adipisicing elit. Delectus, voluptate.
    </div>
</div>
```

Lorem ipsum dolor sit amet, consectetur adipisicing elit. Delectus, voluptate.

```
<div class="card">
    <div class="card-body">
        <h4 class="card-title">Card Title</h4>
        <h6 class="card-subtitle">Card subtitle</h6>
        <p class="card-text">Lorem ipsum dolor sit amet, consectetur adipisicing elit. Illo, quas.</p>
         <a class="btn btn-secondary" href="#">Read More</a>
    </div>
</div>
```

Card Title

Card subtitle

Lorem ipsum dolor sit amet, consectetur adipisicing elit. Illo, quas.

READ MORE

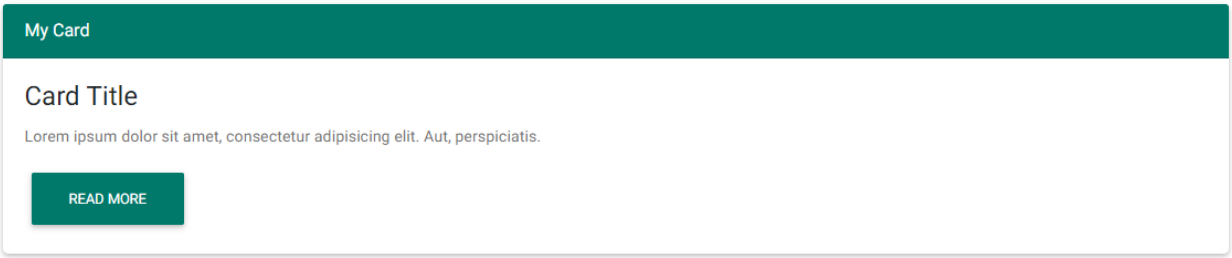
```
<div style="width:20rem" class="card">
    <img class="card-img-top" src="img/card.jpg" alt="Card image cap">
    <div class="card-body">
        <h4 class="card-title">Card Title</h4>
        <p class="card-text">Lorem ipsum dolor sit amet, consectetur adipisicing elit. Illo, quas.</p>
         <a class="btn btn-success" href="#">Read More</a>
    </div>
</div>
```

Card Title

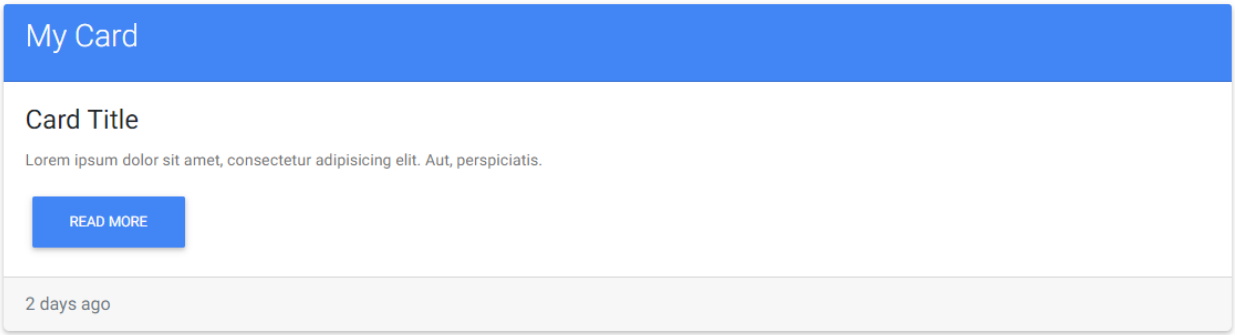
Lorem ipsum dolor sit amet, consectetur adipisicing elit. Illo, quas.

READ MORE

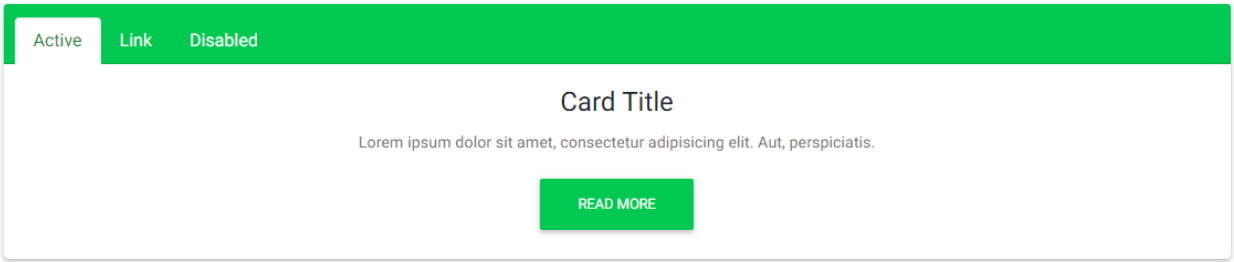
```html
<div class="card">
     <div class="card-header bg-teal text-white">
         My Card
     </div>
     <div class="card-body">
         <h4 class="card-title">Card Title</h4>
         <p class="card-text">Lorem ipsum dolor sit amet, consectetur adipisicing elit.</p>
         <a class="btn btn-teal" href="#">Read More</a>
     </div>
 </div>
```



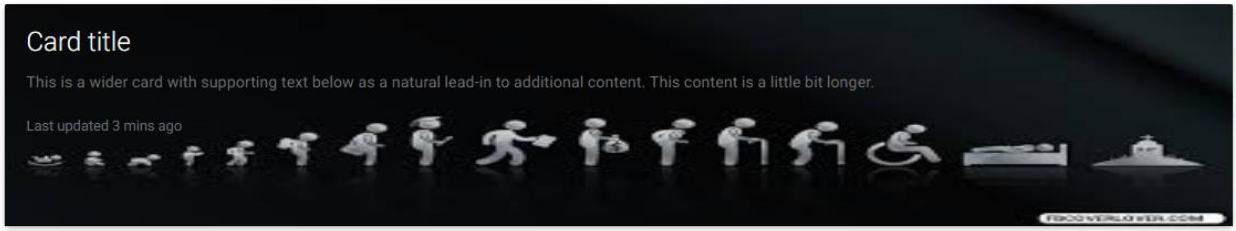

```html
<div class="card">
    <div class="card-header bg-primary text-white">
        <h3 class="card-title">My Card</h3>
    </div>
    <div class="card-body">
        <h4 class="card-title">Card Title</h4>
        <p class="card-text">Lorem ipsum dolor sit amet, consectetur adipisicing elit</p>
        <a class="btn btn-primary" href="#">Read More</a>
    </div>
    <div class="card-footer text-muted">
        2 days ago
    </div>
</div>
```



```
<div class="card text-center">
    <div class="card-header bg-success ">
        <ul class="nav nav-tabs card-header-tabs ">
            <li class="nav-item">
                <a class="nav-link active text-success" href="#">Active</a>
            </li>
            <li class="nav-item">
                <a class="nav-link text-white" href="#">Link</a>
            </li>
            <li class="nav-item">
                <a class="nav-link text-white disabled" href="#">Disabled</a>
            </li>
        </ul>
    </div>
    <div class="card-body">
        <h4 class="card-title">Card Title</h4>
        <p class="card-text">Lorem ipsum dolor sit amet, consectetur adipisicing elit</p>
        <a class="btn btn-success" href="#">Read More</a>
    </div>
</div>
```




```
<div class="card">
    <img class="card-img" src="https://encrypted-tbn0.gstatic.com/images" style="width: auto; height:
200px">
    <div class="card-img-overlay">
        <h4 class="card-title text-white">Card title</h4>
        <p class="card-text">This is a wider card with supporting text below as a natural This content
is a little bit longer.</p>
        <p class="text-muted"><small>Last updated 3 mins ago</small></p>
    </div>
</div>
```




```
<div class="card bg-info text-white">
    <div class="card-body">
        <h4 class="card-title">Card Title</h4>
        <p class="card-text">Lorem ipsum dolor sit amet, consectetur adipisicing elit. Illo, quas.</p>
    </div>
</div>
```

Card Title

Lorem ipsum dolor sit amet, consectetur adipisicing elit. Illo, quas.

16. Media Objects

The media object helps build complex and repetitive components where some media is positioned alongside content that doesn't wrap around said media. Plus, it does this with only two required classes thanks to flexbox.

Below is an example of a single media object. Only two classes are required—the wrapping .media and the .media-body around your content. Optional padding and margin can be controlled through spacing utilities.

```
<div class="media">
    <img class="d-flex mr-2" src="img/bootstrap.png" >
    <div class="media-body">
        <h5>Media heading</h5>
        Cras sit amet nibh libero, in gravida nulla. Nulla vel metus scelerisque ante sollicitudin.
Cras purus odio, vestibulum in vulputate at, tempus viverra turpis. Fusce condimentum nunc ac nisi
vulputate fringilla. Donec lacinia congue felis in faucibus.
    </div>
</div>
```

Media heading

Cras sit amet nibh libero, in gravida nulla. Nulla vel metus scelerisque ante sollicitudin. Cras purus odio, vestibulum in vulputate at, tempus viverra turpis. Fusce condimentum nunc ac nisi vulputate fringilla. Donec lacinia congue felis in faucibus.

```
<div class="media">
    <img src="img/bootstrap.png" class="d-flex mr-2">
    <div class="media-body">
        <h5>Media heading</h5>
        Cras sit amet nibh libero, in gravida nulla. Nulla vel metus scelerisque ante sollicitudin.
Cras purus odio, vestibulum in vulputate at, tempus viverra turpis. Fusce condimentum nunc ac nisi
vulputate fringilla. Donec lacinia congue felis in faucibus.

        <div class="media mt-3">
        <a href="#" class="d-flex mr-2">
            <img src="img/bootstrap.png">
        </a>
        <div class="media-body">
            <h5>Media heading</h5>
            Cras sit amet nibh libero, in gravida nulla. Nulla vel metus scelerisque ante
sollicitudin. Cras purus odio, vestibulum in vulputate at, tempus viverra turpis. Fusce condimentum
nunc ac nisi vulputate fringilla. Donec lacinia congue felis in faucibus.
        </div>
        </div>
    </div>
</div>
```

Media heading

Cras sit amet nibh libero, in gravida nulla. Nulla vel metus scelerisque ante sollicitudin. Cras purus odio, vestibulum in vulputate at, tempus viverra turpis. Fusce condimentum nunc ac nisi vulputate fringilla. Donec lacinia congue felis in faucibus.

Media heading

Cras sit amet nibh libero, in gravida nulla. Nulla vel metus scelerisque ante sollicitudin. Cras purus odio, vestibulum in vulputate at, tempus viverra turpis. Fusce condimentum nunc ac nisi vulputate fringilla. Donec lacinia congue felis in faucibus.

```
<div class="media">
    <img class="d-flex mr-2 align-self-start" src="img/bootstrap.png">
    <div class="media-body">
        <h5>Top aligned media</h5>
        <p>Cras sit amet nibh libero, in gravida nulla. Nulla vel metus scelerisque ante sollicitudin.
Cras purus odio, vestibulum in vulputate at, tempus viverra turpis. Fusce condimentum nunc ac nisi
vulputate fringilla. Donec lacinia congue felis in faucibus.</p>
        <p>Donec sed odio dui. Nullam quis risus eget urna mollis ornare vel eu leo. Cum sociis natoque
penatibus et magnis dis parturient montes, nascetur ridiculus mus.</p>
    </div>
</div>
```

Top aligned media

Cras sit amet nibh libero, in gravida nulla. Nulla vel metus scelerisque ante sollicitudin. Cras purus odio, vestibulum in vulputate at, tempus viverra turpis. Fusce condimentum nunc ac nisi vulputate fringilla. Donec lacinia congue felis in faucibus.

Donec sed odio dui. Nullam quis risus eget urna mollis ornare vel eu leo. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus.

```
<div class="media">
    <img class="d-flex align-self-center mr-2" src="img/bootstrap.png">
    <div class="media-body">
        <h5>Center aligned media</h5>
        <p>Cras sit amet nibh libero, in gravida nulla. Nulla vel metus scelerisque ante sollicitudin.
Cras purus odio, vestibulum in vulputate at, tempus viverra turpis. Fusce condimentum nunc ac nisi
vulputate fringilla. Donec lacinia congue felis in faucibus.</p>
        <p>Donec sed odio dui. Nullam quis risus eget urna mollis ornare vel eu leo. Cum sociis natoque
penatibus et magnis dis parturient montes, nascetur ridiculus mus.</p>
    </div>
</div>
```

Center aligned media

Cras sit amet nibh libero, in gravida nulla. Nulla vel metus scelerisque ante sollicitudin. Cras purus odio, vestibulum in vulputate at, tempus viverra turpis. Fusce condimentum nunc ac nisi vulputate fringilla. Donec lacinia congue felis in faucibus.

Donec sed odio dui. Nullam quis risus eget urna mollis ornare vel eu leo. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus.

```
<div class="media">
    <img class="d-flex align-self-end mr-2" src="img/bootstrap.png">
    <div class="media-body">
        <h5>Bottom aligned media</h5>
        <p>Cras sit amet nibh libero, in gravida nulla. Nulla vel metus scelerisque ante sollicitudin.
Cras purus odio, vestibulum in vulputate at, tempus viverra turpis. Fusce condimentum nunc ac nisi
vulputate fringilla. Donec lacinia congue felis in faucibus.</p>
        <p>Donec sed odio dui. Nullam quis risus eget urna mollis ornare vel eu leo. Cum sociis natoque
penatibus et magnis dis parturient montes, nascetur ridiculus mus.</p>
    </div>
</div>
```

Bottom aligned media

Cras sit amet nibh libero, in gravida nulla. Nulla vel metus scelerisque ante sollicitudin. Cras purus odio, vestibulum in vulputate at, tempus viverra turpis. Fusce condimentum nunc ac nisi vulputate fringilla. Donec lacinia congue felis in faucibus.

Donec sed odio dui. Nullam quis risus eget urna mollis ornare vel eu leo. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus.

```
<ul class="list-unstyled">
    <li class="media bg-success p-3 text-white">
        <img src="img/bootstrap.png" class="d-flex mr-2">
        <div class="media-body">
        <h5>List based media object</h5>
        Cras sit amet nibh libero, in gravida nulla. Nulla vel metus scelerisque ante sollicitudin.
Cras purus odio, vestibulum in vulputate at, tempus viverra turpis. Fusce condimentum nunc ac nisi
vulputate fringilla. Donec lacinia congue felis in faucibus.
        </div>
    </li>
    <li class="media bg-warning p-3 text-white">
        <img class="d-flex mr-2" src="img/bootstrap.png">
        <div class="media-body">
        <h5>List based media object</h5>
        Cras sit amet nibh libero, in gravida nulla. Nulla vel metus scelerisque ante sollicitudin.
Cras purus odio, vestibulum in vulputate at, tempus viverra turpis. Fusce condimentum nunc ac nisi
vulputate fringilla. Donec lacinia congue felis in faucibus.
        </div>
    </li>
</ul>
```

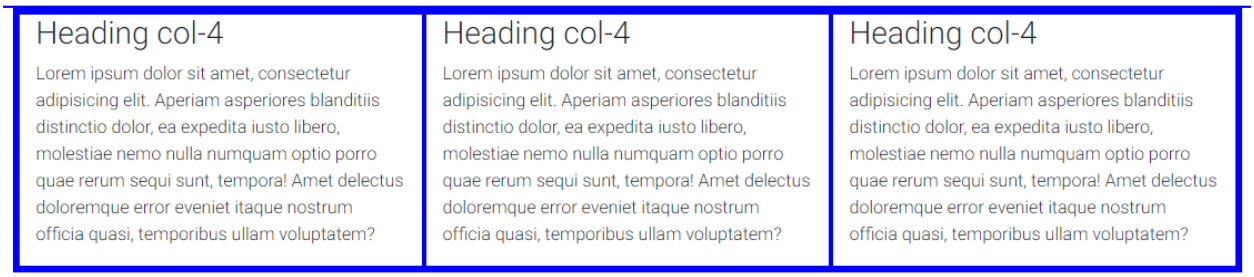
17. Grid System in Bootstrap

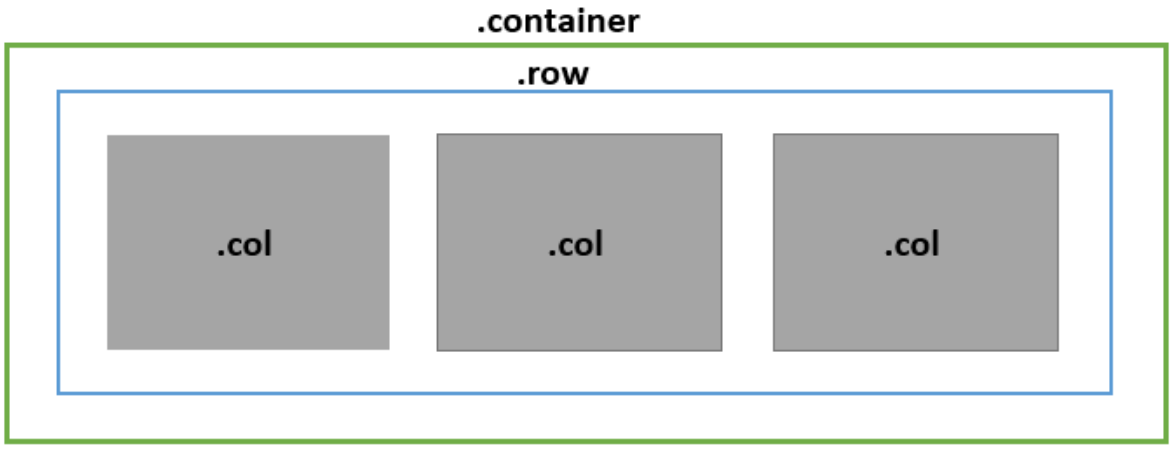
Bootstrap grid system comes with 12 column layout built with flexboxes to allow responsiveness.

The content should be within the container for fixed width or container-fluid for full width pages.

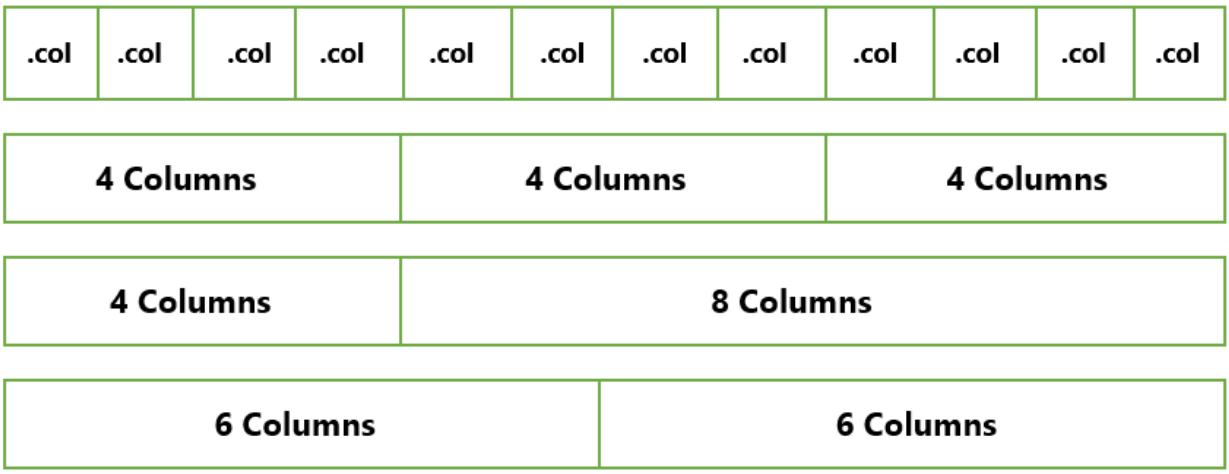
We should use '.row' class to lined up the content in one line, and the content should be inside a '.col' class.

```
<div class="container">
    <div class="row">
        <div class='col-4'>
            <h3>Heading col-4</h3>
            <p>Lorem ipsum dolor sit amet, consectetur adipisicing elit. Aperiam asperiores blanditiis
distinctio dolor</p>
        </div>
        <div class='col-4'>
            <h3>Heading col-4</h3>
            <p>Lorem ipsum dolor sit amet, consectetur adipisicing elit. Aperiam asperiores blanditiis
distinctio dolor</p>
        </div>
        <div class='col-4'>
            <h3>Heading col-4</h3>
            <p>Lorem ipsum dolor sit amet, consectetur adipisicing elit. Aperiam asperiores blanditiis
distinctio dolor</p>
        </div>
    </div>
</div>
```





In Bootstrap the width of the browser is divided into 12 col layout.

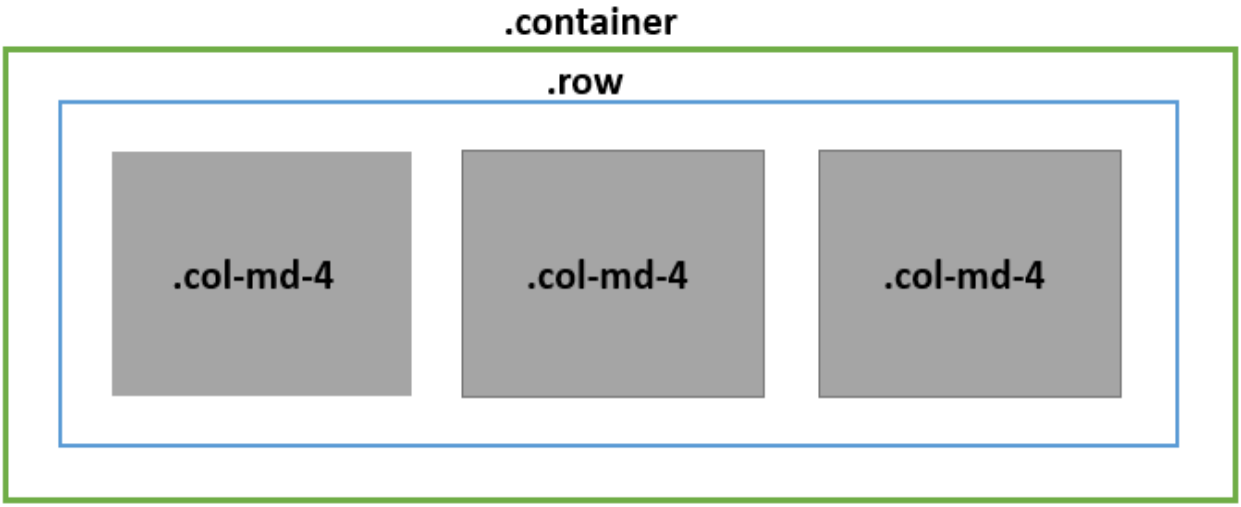


Responsive classes

Bootstrap comes with responsive classes such as,

$$\text{Col} - \{\text{sm / md / lg / xl}\} - \{\text{number}\}$$

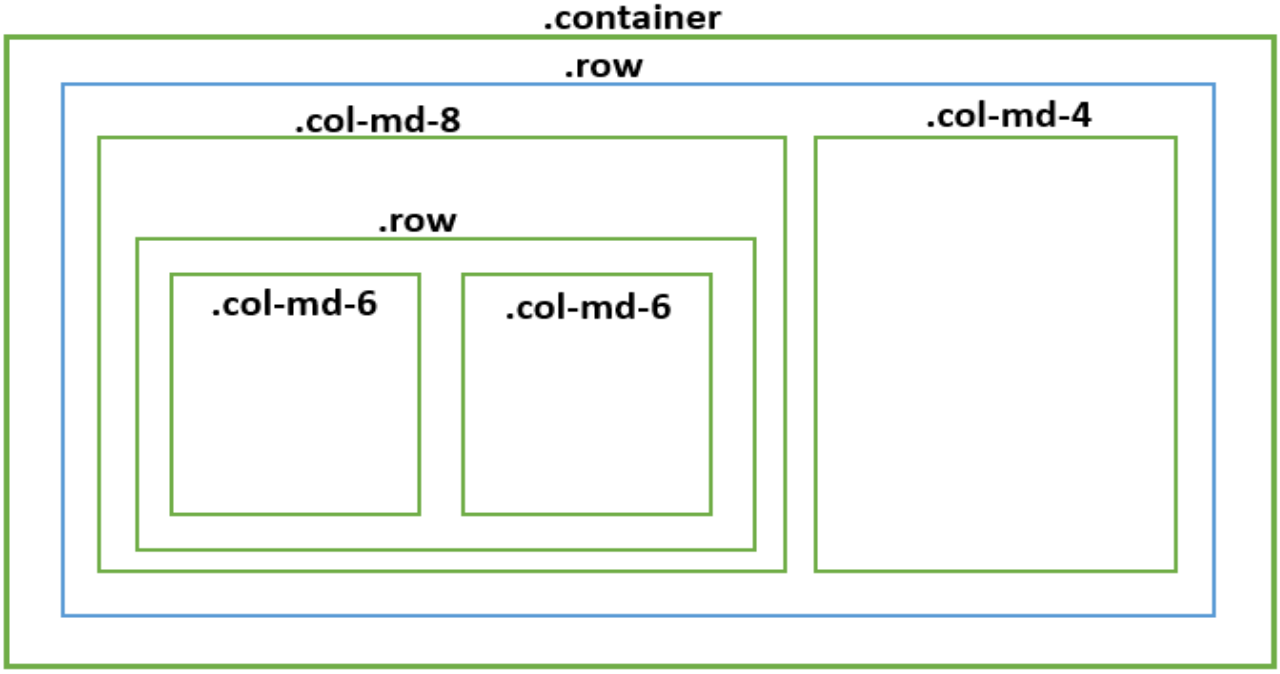
To display 3-cols in medium screen,



```
<div class="container">
        <div class="row">
            <div class="col-md-4">
                ....
            </div>
            <div class="col-md-4">
                ....
            </div>
            <div class="col-md-4">
                ....
            </div>
        </div>
</div>
```

Nesting Grids

We can even do the nesting of grid elements on inside the other. Each nested grid also have the width of 12 column layout.



```
<!-- NESTING GRIDS -->
<div class="container">
    <div class="row">
        <div class="col-md-8">
                .......
            <div class="row">
                <div class="col-md-6">
                    ......
                </div>
                <div class="col-md-6">
                    ......
                </div>
            </div>
        </div>
        <div class="col-md-4">
            ......
        </div>
    </div>
</div>
```

18. Grid Alignments

Bootstrap uses flexbox alignment utilities to align the columns vertically or horizontally in the grid.

Vertical Alignment

The following are the vertical alignment classes

.*align-items-start* -> to align all the items in the grid to start of the grid

. *align-items-center* -> to align all the items in the grid to center of the grid

. *align-items-end* -> to align all the items in the grid to end of the grid

We can even align individual columns using the following classes.

. *align-self-start* -> to align to the start of the grid.

. *align-self-center* -> to align to the center of the grid

. *align-self-end* -> to align to the end of the grid

Horizontal alignment

The following Bootstrap classes are used to align items horizontally.

. *justify-content-start* -> to align all the items to left side of the grid.

. *justify-content-center* -> to align all the items to center of the grid.

. *justify-content-end* -> to align all the items to end of the grid.

. *justify-content-around* -> to align the items in the grid with some space around them.

. *justify-content-between* -> to align the items in the grid with some space between them.

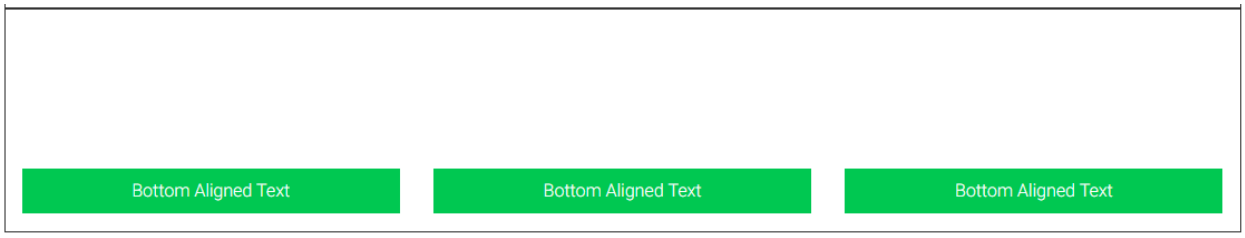
```html
<div class="row align-items-start" style="height:200px;border:1px #333 solid">
    <div class="col">
        <p class="bg-success p-2 text-white text-center">Top Aligned Text</p>
    </div>
    <div class="col">
        <p class="bg-success p-2 text-white text-center">Top Aligned Text</p>
    </div>
    <div class="col">
        <p class="bg-success p-2 text-white text-center">Top Aligned Text</p>
    </div>
</div>
```

Top Aligned Text | Top Aligned Text | Top Aligned Text

```html
<div class="row align-items-center" style="height:200px;border:1px #333 solid">
    <div class="col">
        <p class="bg-success p-2 text-white text-center">Middle Aligned Text</p>
    </div>
    <div class="col">
        <p class="bg-success p-2 text-white text-center">Middle Aligned Text</p>
    </div>
    <div class="col">
        <p class="bg-success p-2 text-white text-center">Middle Aligned Text</p>
    </div>
</div>
```

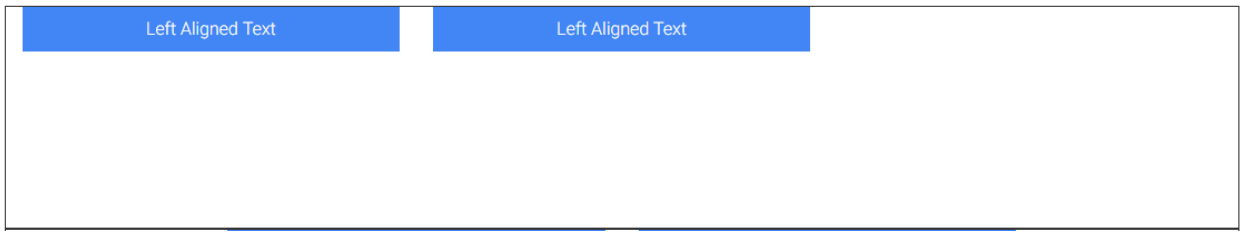
Middle Aligned Text | Middle Aligned Text | Middle Aligned Text

```html
<div class="row align-items-end" style="height:200px;border:1px #333 solid">
    <div class="col">
        <p class="bg-success p-2 text-white text-center">Bottom Aligned Text</p>
    </div>
    <div class="col">
        <p class="bg-success p-2 text-white text-center">Bottom Aligned Text</p>
    </div>
    <div class="col">
        <p class="bg-success p-2 text-white text-center">Bottom Aligned Text</p>
    </div>
</div>
```




```
<div class="container">
    <div class="row" style="height:200px;border:1px #333 solid">
        <div class="col align-self-start">
            <p class="bg-warning text-center text-white p-2">Top Aligned</p>
        </div>
        <div class="col align-self-center">
            <p class="bg-warning text-center text-white p-2">Middle Aligned</p>
        </div>
        <div class="col align-self-end">
            <p class="bg-warning text-center text-white p-2">Bottom Aligned</p>
        </div>
    </div>
</div>
```

```
<div class="row justify-content-start" style="height:200px;border:1px #333 solid">
    <div class="col-4">
        <p class="bg-primary text-white text-center p-2">Left Aligned Text</p>
    </div>
    <div class="col-4">
        <p class="bg-primary text-white text-center p-2">Left Aligned Text</p>
    </div>
</div>
```



```
<div class="row justify-content-center" style="height:200px;border:1px #333 solid">
    <div class="col-4">
        <p class="bg-primary text-white text-center p-2">Left Aligned Text</p>
    </div>
    <div class="col-4">
        <p class="bg-primary text-white text-center p-2">Left Aligned Text</p>
    </div>
</div>
```

Left Aligned Text

Left Aligned Text

```
<div class="row justify-content-end" style="height:200px;border:1px #333 solid">
    <div class="col-4">
        <p class="bg-primary text-white text-center p-2">Left Aligned Text</p>
    </div>
    <div class="col-4">
        <p class="bg-primary text-white text-center p-2">Left Aligned Text</p>
    </div>
</div>
```

Left Aligned Text

Left Aligned Text

```
<div class="row justify-content-around" style="height:200px;border:1px #333 solid">
    <div class="col-4">
        <p class="bg-primary text-white text-center p-2">Left Aligned Text</p>
    </div>
    <div class="col-4">
        <p class="bg-primary text-white text-center p-2">Left Aligned Text</p>
    </div>
</div>
```

Left Aligned Text

Left Aligned Text

```html
<div class="row justify-content-between" style="height:200px;border:1px #333 solid">
    <div class="col-4">
        <p class="bg-primary text-white text-center p-2">Left Aligned Text</p>
    </div>
    <div class="col-4">
        <p class="bg-primary text-white text-center p-2">Left Aligned Text</p>
    </div>
</div>
```

Left Aligned Text

Left Aligned Text

```html
<div class="row no-gutters">
        <div class="col-4">
            <div class="card">
                <div class="card-body">
                    <h3 class="card-title">Card Title</h3>
                    <p class="card-text">Lorem</p>
                </div>
            </div>
        </div>
        <div class="col-4">
            <div class="card">
                <div class="card-body">
                    <h3 class="card-title">Card Title</h3>
                    <p class="card-text">Lorem </p>
                </div>
            </div>
        </div>
        <div class="col-4">
            <div class="card">
                <div class="card-body">
                    <h3 class="card-title">Card Title</h3>
                    <p class="card-text">Lorem </p>
                </div>
            </div>
        </div>
    </div>
```

Card Title

Lorem ipsum dolor sit amet, consectetur adipisicing elit. Accusamus architecto consequatur deserunt dicta doloribus ea eaque eveniet, ex facilis harum nulla odio placeat quaerat qui sequi sit voluptatum? Eaque, esse!

Card Title

Lorem ipsum dolor sit amet, consectetur adipisicing elit. Accusamus architecto consequatur deserunt dicta doloribus ea eaque eveniet, ex facilis harum nulla odio placeat quaerat qui sequi sit voluptatum? Eaque, esse!

Card Title

Lorem ipsum dolor sit amet, consectetur adipisicing elit. Accusamus architecto consequatur deserunt dicta doloribus ea eaque eveniet, ex facilis harum nulla odio placeat quaerat qui sequi sit voluptatum? Eaque, esse!

19. Flex Box

Bootstrap Flexbox classes are used to quickly manage the layout, alignment, and sizing of grid columns, navigation, components, and more with a full suite of responsive flexbox utilities. For more complex implementations, custom CSS may be necessary.
To create a flex-box container , we have to use a base class called 'd-flex'

```
<div class="d-flex p-2">I'm a flexbox container!</div>
```

Direction

We can set the direction of flex-items in the flex container with the direction utilities. In most cases you can omit the horizontal class here as the browser default is row. However, you may encounter situations where you needed to explicitly set this value (like responsive layouts).
Use .flex-row to set a horizontal direction (the browser default), or .flex-row-reverse to start the horizontal direction from the opposite side.

```
<h3>FLEX ROW & FLEX ITEMS</h3>
<div class="row-hl d-flex flex-row">
    <div class="item-hl p-1 display-4">1</div>
    <div class="item-hl p-1 display-4">2</div>
    <div class="item-hl p-1 display-4">3</div>
</div>
```

FLEX ROW & FLEX ITEMS

```
<h3>FLEX ROW REVERSE</h3>
<div class="row-hl d-flex flex-row-reverse">
    <div class="item-hl p-1 display-4">1</div>
    <div class="item-hl p-1 display-4">2</div>
    <div class="item-hl p-1 display-4">3</div>
</div>
```

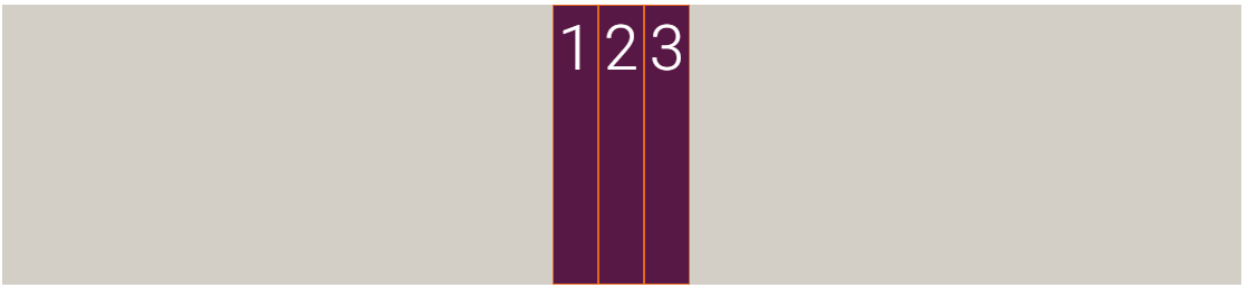
FLEX ROW REVERSE

```
<h3>JUSTIFY CONTENT START</h3>
<div class="row-hl d-flex flex-row justify-content-start">
    <div class="item-hl p-1 display-4">1</div>
    <div class="item-hl p-1 display-4">2</div>
    <div class="item-hl p-1 display-4">3</div>
</div>
```

JUSTIFY CONTENT START

```
<h3>JUSTIFY CONTENT CENTER</h3>
<div class="row-hl d-flex flex-row justify-content-center">
    <div class="item-hl p-1 display-4">1</div>
    <div class="item-hl p-1 display-4">2</div>
    <div class="item-hl p-1 display-4">3</div>
</div>
```

JUSTIFY CONTENT CENTER




```
<h3>JUSTIFY CONTENT END</h3>
<div class="row-hl d-flex flex-row justify-content-end">
    <div class="item-hl p-1 display-4">1</div>
    <div class="item-hl p-1 display-4">2</div>
    <div class="item-hl p-1 display-4">3</div>
</div>
```

JUSTIFY CONTENT END

```
<h3>JUSTIFY CONTENT AROUND</h3>
<div class="row-hl d-flex flex-row justify-content-around">
    <div class="item-hl p-1 display-4">1</div>
    <div class="item-hl p-1 display-4">2</div>
    <div class="item-hl p-1 display-4">3</div>
</div>
```

JUSTIFY CONTENT AROUND

```
<h3>JUSTIFY CONTENT BETWEEN</h3>
<div class="row-hl d-flex flex-row justify-content-between">
    <div class="item-hl p-1 display-4">1</div>
    <div class="item-hl p-1 display-4">2</div>
    <div class="item-hl p-1 display-4">3</div>
</div>
```

JUSTIFY CONTENT BETWEEN

```
<h3>FLEX CONTAINER / FLEX COLUMN</h3>
<div class="row-hl d-flex flex-column">
    <div class="item-hl p-1 display-4">1</div>
    <div class="item-hl p-1 display-4">2</div>
    <div class="item-hl p-1 display-4">3</div>
</div>
```

FLEX CONTAINER / FLEX COLUMN

1

2

3

```
<h3>FLEX COLUMN REVERSE</h3>
<div class="row-hl d-flex flex-column-reverse">
    <div class="item-hl p-1 display-4">1</div>
    <div class="item-hl p-1 display-4">2</div>
    <div class="item-hl p-1 display-4">3</div>
</div>
```

FLEX COLUMN REVERSE

3

2

1

```
<h3>VERTICAL ALIGN ITEMS START</h3>
<div class="row-hl d-flex flex-column align-items-start">
    <div class="item-hl p-1 display-4">1</div>
    <div class="item-hl p-1 display-4">2</div>
    <div class="item-hl p-1 display-4">3</div>
</div>
```

VERTICAL ALIGN ITEMS START

1

2

3

```
<div class="row-hl d-flex flex-column align-items-center">
    <div class="item-hl p-1 display-4">1</div>
    <div class="item-hl p-1 display-4">2</div>
    <div class="item-hl p-1 display-4">3</div>
</div>
```

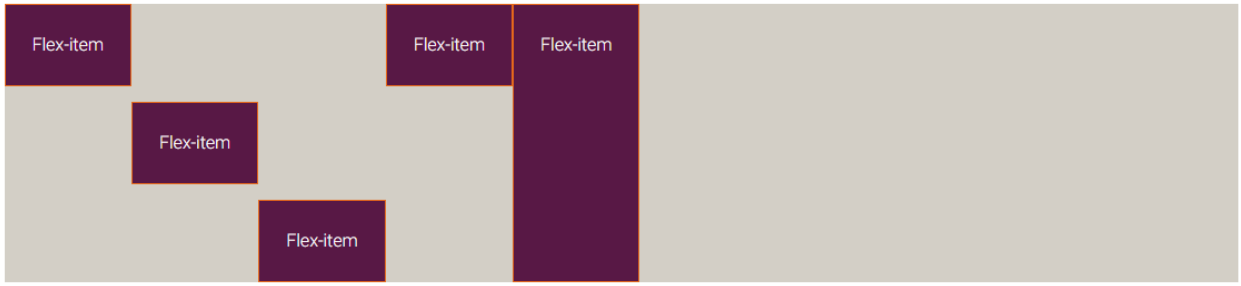
VERTICAL ALIGN ITEMS CENTER

```
<h3>VERTICAL ALIGN ITEMS END</h3>
<div class="row-hl d-flex flex-column align-items-end">
    <div class="item-hl p-1 display-4">1</div>
    <div class="item-hl p-1 display-4">2</div>
    <div class="item-hl p-1 display-4">3</div>
</div>
```

VERTICAL ALIGN ITEMS END

```
<h3>ALIGN SELF</h3>
<div class="row-hl d-flex flex-row">
    <div class="item-hl p-4 align-self-start">Flex-item</div>
    <div class="item-hl p-4  align-self-center">Flex-item</div>
    <div class="item-hl p-4  align-self-end">Flex-item</div>
    <div class="item-hl p-4  align-self-baseline">Flex-item</div>
    <div class="item-hl p-4  align-self-stretch">Flex-item</div>
</div>
```

ALIGN SELF



20. Auto Margins

Flexbox can do some pretty awesome things when you mix flex alignments with auto margins. Shown below are three examples of controlling flex items via auto margins: default (no auto margin), pushing two items to the right (.mr-auto), and pushing two items to the left (.ml-auto).

Note: **Unfortunately, IE10 and IE11 do not properly support auto margins on flex items whose parent has a non-default justify-content value.**

```
<div class="d-flex row-hl flex-row">
    <div class="p-2 item-hl">Flex item</div>
    <div class="p-2 item-hl mr-auto">Flex item</div>
    <div class="p-2 item-hl">Flex item</div>
</div>
```

```
<!-- ML-AUTO -->
<div class="d-flex row-hl flex-row">
    <div class="p-2 item-hl">Flex item</div>
    <div class="p-2 item-hl ml-auto">Flex item</div>
    <div class="p-2 item-hl ">Flex item</div>
</div>
```

With Align Items

Vertically move one flex item to the top or bottom of a container by mixing align-items, flex-direction: column, and margin-top: auto or margin-bottom: auto.

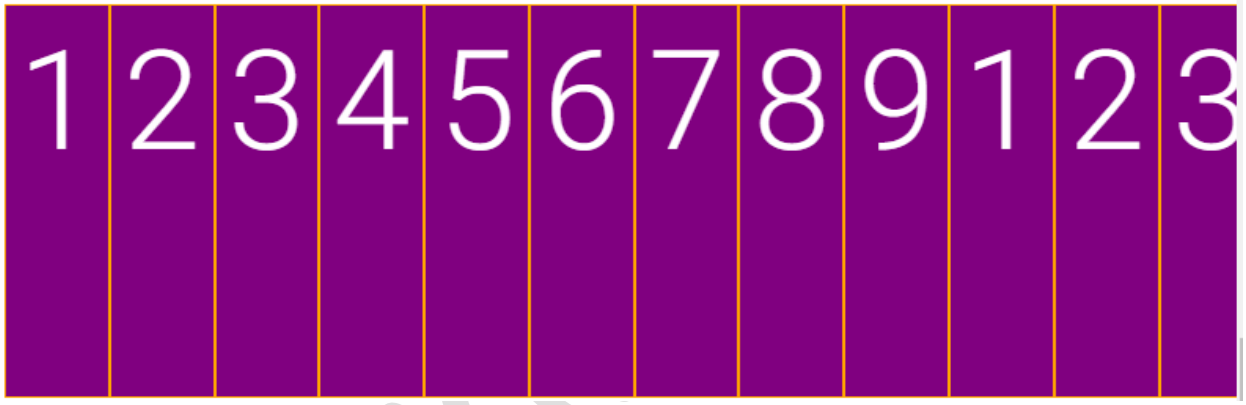
```html
<!-- MB-AUTO -->
<div class="d-flex row-hl flex-column">
    <div class="p-2 item-hl">Flex item</div>
    <div class="p-2 item-hl mb-auto">Flex item</div>
    <div class="p-2 item-hl">Flex item</div>
</div>
```

```html
<!-- MT-AUTO -->
<div class="d-flex row-hl flex-column">
    <div class="p-2 item-hl">Flex item</div>
    <div class="p-2 item-hl mt-auto">Flex item</div>
    <div class="p-2 item-hl">Flex item</div>
</div>
```

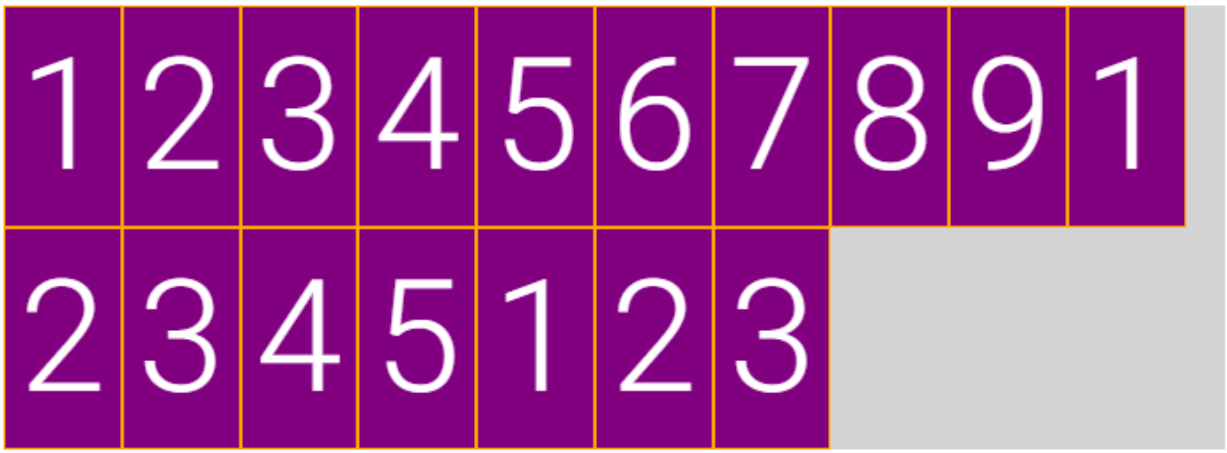
Wrapping

Change how flex items wrap in a flex container. Choose from no wrapping at all (the browser default) with .flex-nowrap, wrapping with .flex-wrap, or reverse wrapping with .flex-wrap-reverse.

```html
<!-- NO WRAP -->
<div class="d-flex row-hl flex-row flex-nowrap">
        <div class="p-2 item-hl display-2">1</div>
        <div class="p-2 item-hl display-2">2</div>
        <div class="p-2 item-hl display-2">3</div>
        <div class="p-2 item-hl display-2">4</div>
        <div class="p-2 item-hl display-2">5</div>
        <div class="p-2 item-hl display-2">6</div>
        <div class="p-2 item-hl display-2">7</div>
        <div class="p-2 item-hl display-2">8</div>
</div>
```




```html
<!-- WRAP -->
    <div class="d-flex row-hl flex-row flex-wrap">
        <div class="p-2 item-hl display-2">1</div>
        <div class="p-2 item-hl display-2">2</div>
        <div class="p-2 item-hl display-2">3</div>
        <div class="p-2 item-hl display-2">4</div>
        <div class="p-2 item-hl display-2">5</div>
        <div class="p-2 item-hl display-2">6</div>
        <div class="p-2 item-hl display-2">7</div>
        <div class="p-2 item-hl display-2">8</div>
        <div class="p-2 item-hl display-2">9</div>
</div>
```



21. Carousel

A slideshow component for cycling through elements—images or slides of text—like a carousel.

The carousel is a slideshow for cycling through a series of content, built with CSS 3D transforms and a bit of JavaScript. It works with a series of images, text, or custom markup. It also includes support for previous/next controls and indicators.

Be sure to set a unique id on the .carousel for optional controls, especially if you're using multiple carousels on a single page.

Here's a carousel with slides only. Note the presence of the .d-block and .img-fluid on carousel images to prevent browser default image alignment.

Simple Carousel

```
<h3>SIMPLE SLIDER</h3>
<div class="carousel slide" id="simpleSlider" data-ride="carousel">
    <div class="carousel-inner">
        <div class="carousel-item active">
            <img src="img/slider1.jpg" class="w-100 d-block">
        </div>
        <div class="carousel-item">
            <img src="img/slider2.jpg" class="w-100 d-block">
        </div>
        <div class="carousel-item">
            <img src="img/slider3.jpg" class="w-100 d-block">
        </div>
    </div>
</div>
```

SIMPLE SLIDER

Carousel with controls

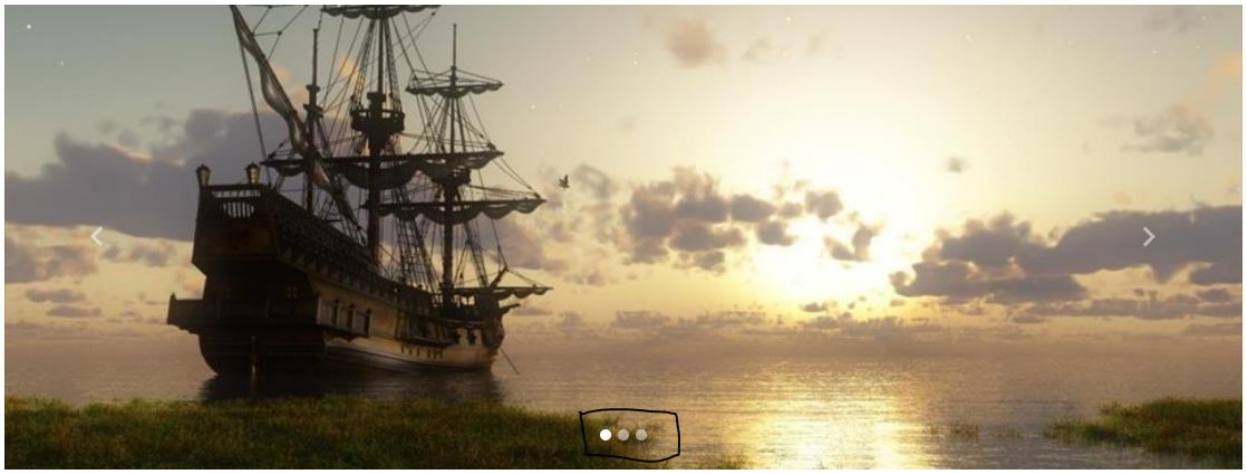
```
<h3>SLIDER WITH CONTROLS</h3>
<div class="carousel slide" id="controlSlider" data-ride="carousel">
    <div class="carousel-inner">
        <div class="carousel-item active">
            <img src="img/slider1.jpg" class="w-100 d-block">
        </div>
        <div class="carousel-item">
            <img src="img/slider2.jpg" class="w-100 d-block">
        </div>
        <div class="carousel-item">
            <img src="img/slider3.jpg" class="w-100 d-block">
        </div>
    </div>
    <a href="#controlSlider" class="carousel-control-prev" data-slide="prev">
        <span class="carousel-control-prev-icon"></span>
    </a>
    <a href="#controlSlider" class="carousel-control-next" data-slide="next">
        <span class="carousel-control-next-icon"></span>
    </a>
</div>
```

SLIDER WITH CONTROLS

Carousel with Indicators

```
<h3>SLIDER WITH INDICATORS</h3>
    <div class="carousel slide" id="indicatorSlider" data-ride="carousel">
        <ol class="carousel-indicators">
            <li data-target="#indicatorSlider"
                data-slide-to="0" class="active"></li>

            <li data-target="#indicatorSlider"
                data-slide-to="1"></li>

            <li data-target="#indicatorSlider"
                data-slide-to="2"></li>
        </ol>

       <div class="carousel-inner">
           <div class="carousel-item active">
               <img src="img/slider1.jpg" class="w-100 d-block">
           </div>
           <div class="carousel-item">
               <img src="img/slider2.jpg" class="w-100 d-block">
           </div>
           <div class="carousel-item">
               <img src="img/slider3.jpg" class="w-100 d-block">
           </div>
       </div>
       <a href="#indicatorSlider" class="carousel-control-prev" data-slide="prev">
           <span class="carousel-control-prev-icon"></span>
       </a>
       <a href="#indicatorSlider" class="carousel-control-next" data-slide="next">
           <span class="carousel-control-next-icon"></span>
       </a>
   </div>
```

SLIDER WITH INDICATORS

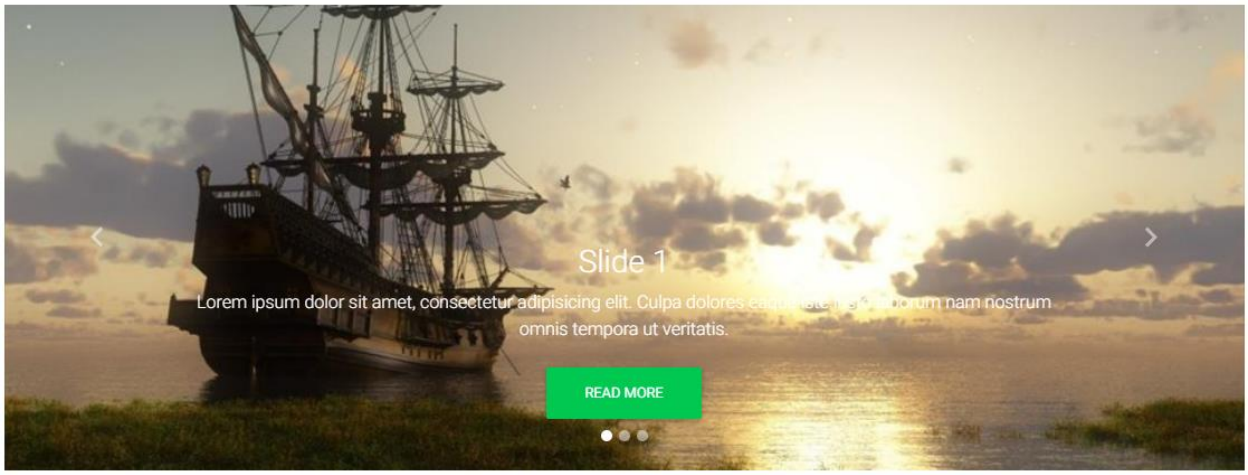


Carousel with Captions

```
<h3>SLIDER WITH CAPTIONS</h3>
<div class="carousel slide" id="captionSlider" data-ride="carousel">
    <ol class="carousel-indicators">
        <li data-target="#captionSlider"
            data-slide-to="0" class="active"></li>

        <li data-target="#captionSlider"
            data-slide-to="1"></li>

        <li data-target="#captionSlider"
            data-slide-to="2"></li>
    </ol>

    <div class="carousel-inner">
        <div class="carousel-item active">
            <img src="img/slider1.jpg" class="w-100 d-block">
            <div class="carousel-caption">
                <h3>Slide 1</h3>
                <p>Lorem ipsum dolor sit amet, consectetur adipisicing elit</p>
                <a class="btn btn-success">Read More</a>
            </div>
        </div>
        <div class="carousel-item">
            <img src="img/slider2.jpg" class="w-100 d-block">
            <div class="carousel-caption">
                <h3>Slide 2</h3>
                <p>Lorem ipsum dolor sit amet, consectetur adipisicing elit</p>
                <a class="btn btn-warning">Read More</a>
            </div>
        </div>
```

```
        <div class="carousel-item">
            <img src="img/slider3.jpg" class="w-100 d-block">
            <div class="carousel-caption">
                <h3>Slide 3</h3>
                <p>Lorem ipsum dolor sit amet, consectetur adipisicing elit</p>
                <a class="btn btn-danger">Read More</a>
            </div>
        </div>
    </div>
    <a href="#captionSlider" class="carousel-control-prev" data-slide="prev">
        <span class="carousel-control-prev-icon"></span>
    </a>
    <a href="#captionSlider" class="carousel-control-next" data-slide="next">
        <span class="carousel-control-next-icon"></span>
    </a>
</div>
```

SLIDER WITH CAPTIONS



22. Collapse

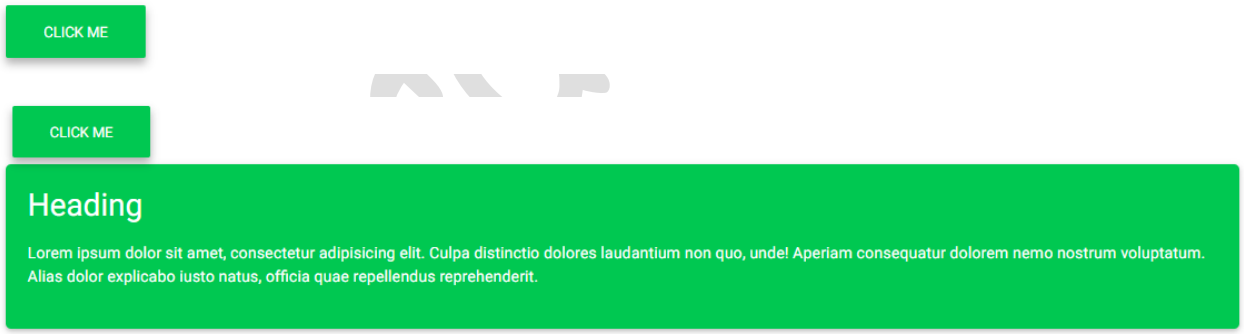
Toggle the visibility of content across your project with a few classes and our JavaScript plugins.

- .collapse hides content
- .collapsing is applied during transitions
- .collapse.show shows content

You can use a link with the href attribute, or a button with the data-target attribute. In both cases, the data-toggle="collapse" is required.

Button with Collapse

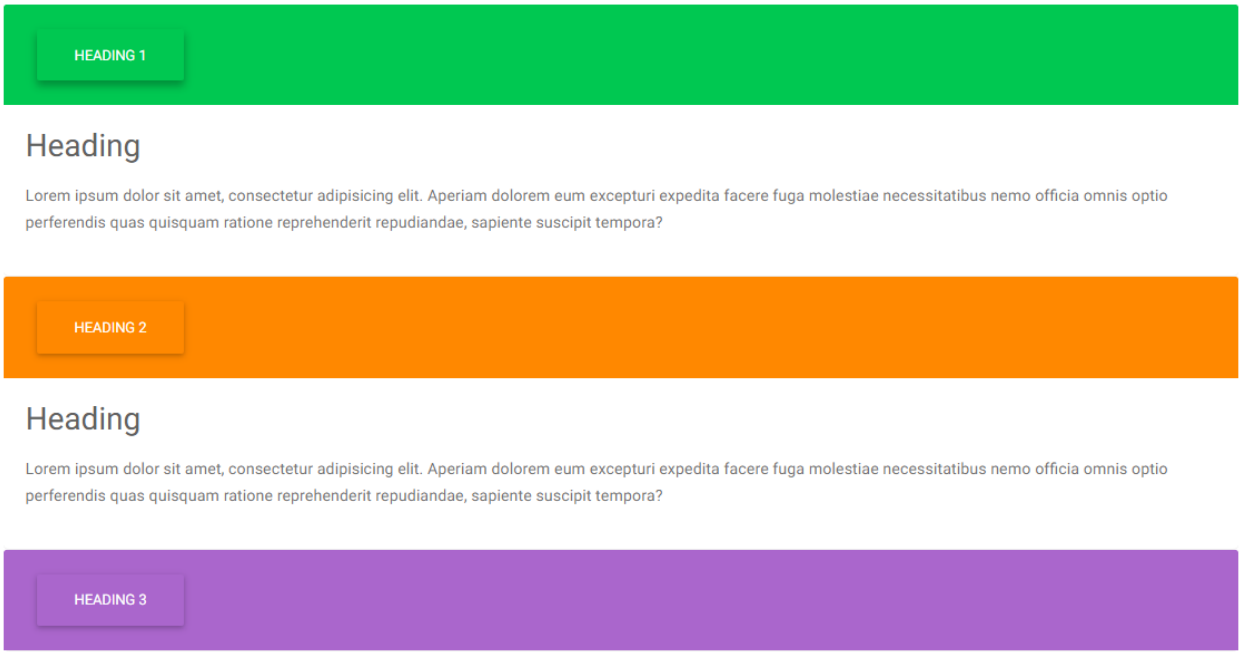
```
<!-- BUTTON WITH COLLAPSE -->
<a class="btn btn-success" data-toggle="collapse"
   href="#btnCollapse">Click Me</a>

<div class="collapse" id="btnCollapse">
    <div class="card bg-success text-white">
        <div class="card-body">
            <h3 class="card-title">Heading</h3>
            <p class="card-text">Lorem ipsum dolor sit amet, consectetur adipisicing elit. Culpa
distinctio dolores laudantium non quo, unde! Aperiam consequatur dolorem nemo nostrum voluptatum. Alias
dolor explicabo iusto natus, officia quae repellendus reprehenderit.</p>
        </div>
    </div>
</div>
```



Accordion

Using the card component, you can extend the default collapse behavior to create an accordion.

```html
<div class="accordion mt-3">
    <div class="card">
        <div class="card-header bg-success">
            <a class="btn btn-success"
               data-toggle="collapse" href="#heading1">Heading 1</a>
        </div>
        <div class="collapse" id="heading1">
            <div class="card-body">
                <h3 class="card-title">Heading</h3>
                <p class="card-text">Lorem ipsum dolor sit amet, consectetur adipisicing elit. Aperiam
dolorem eum excepturi expedita facere fuga molestiae necessitatibus nemo officia omnis optio
perferendis quas quisquam ratione reprehenderit repudiandae, sapiente suscipit tempora?</p>
            </div>
        </div>
    </div>
    <div class="card">
        <div class="card-header bg-warning">
            <a class="btn btn-warning" data-toggle="collapse" href="#heading2">Heading 2</a>
        </div>
        <div class="collapse" id="heading2">
            <div class="card-body">
                <h3 class="card-title">Heading</h3>
                <p class="card-text">Lorem ipsum dolor sit amet, consectetur adipisicing elit. Aperiam
dolorem eum excepturi expedita facere fuga molestiae necessitatibus nemo officia omnis optio
perferendis quas quisquam ratione reprehenderit repudiandae, sapiente suscipit tempora?</p>
            </div>
        </div>
    </div>
    <div class="card">
        <div class="card-header bg-secondary">
            <a class="btn btn-secondary" data-toggle="collapse" href="#heading3">Heading 3</a>
        </div>
        <div class="collapse" id="heading3">
            <div class="card-body">
                <h3 class="card-title">Heading</h3>
                <p class="card-text">Lorem ipsum dolor sit amet, consectetur adipisicing elit. Aperiam
dolorem eum excepturi expedita facere fuga molestiae necessitatibus nemo officia omnis optio
perferendis quas quisquam ratione reprehenderit repudiandae, sapiente suscipit tempora?</p>
            </div>
        </div>

    </div>
</div>
```



23. Tooltips

Bootstrap comes with tooltips with a third party plugin called 'popper.js' for positioning.

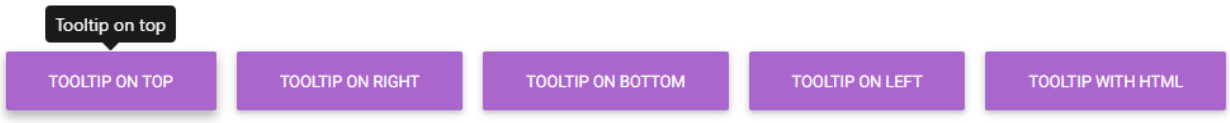
- Tooltips rely on the 3rd party library Popper.js for positioning. You must include popper.min.js before bootstrap.js or use bootstrap.bundle.min.js / bootstrap.bundle.jswhich contains Popper.js in order for tooltips to work!
- If you're building our JavaScript from source, it requires util.js.
- Tooltips are opt-in for performance reasons, so **you must initialize them yourself**.
- Tooltips with zero-length titles are never displayed.

To Enable tooltip, we must use the following snippet,

```
$(function () {
  $('[data-toggle="tooltip"]').tooltip()
})
```

Tooltips Examples

```
<!-- TOOLTIPS -->
<button type="button" class="btn btn-secondary" data-toggle="tooltip" data-placement="top"
title="Tooltip on top">
    Tooltip on top
</button>
<button type="button" class="btn btn-secondary" data-toggle="tooltip" data-placement="right"
title="Tooltip on right">
    Tooltip on right
</button>
<button type="button" class="btn btn-secondary" data-toggle="tooltip" data-placement="bottom"
title="Tooltip on bottom">
    Tooltip on bottom
</button>
<button type="button" class="btn btn-secondary" data-toggle="tooltip" data-placement="left"
title="Tooltip on left">
    Tooltip on left
</button>
<button type="button" class="btn btn-secondary" data-toggle="tooltip" data-html="true" title="
<em>Tooltip</em> <u>with</u> <b>HTML</b>">
    Tooltip with HTML
</button>
```



24. Popovers

- Popovers rely on the 3rd party library Popper.js for positioning. You must include popper.min.js before bootstrap.js or use bootstrap.bundle.min.js / bootstrap.bundle.jswhich contains Popper.js in order for popovers to work!
- Popovers require the tooltip plugin as a dependency.
- If you're building our JavaScript from source, it requires util.js.
- Popovers are opt-in for performance reasons, so **you must initialize them yourself**.

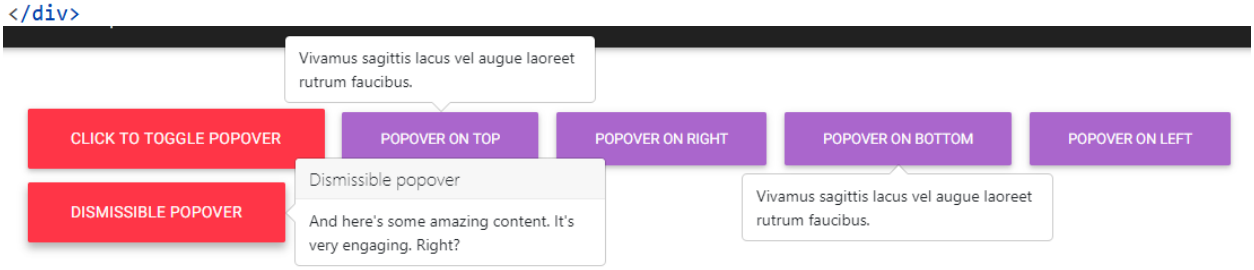
To Enable popovers,

One way to initialize all popovers on a page would be to select them by their data-toggle attribute:

```
$(function () {
  $('[data-toggle="popover"]').popover()
})
```

Popovers Example

```
<!-- POPOVER -->
  <button type="button" class="btn btn-lg btn-danger" data-toggle="popover" title="Popover title"
data-content="And here's some amazing content. It's very engaging. Right?">Click to toggle
popover</button>

  <button type="button" class="btn btn-secondary" data-container="body" data-toggle="popover" data-
placement="top" data-content="Vivamus sagittis lacus vel augue laoreet rutrum faucibus.">
      Popover on top
  </button>

  <button type="button" class="btn btn-secondary" data-container="body" data-toggle="popover" data-
placement="right" data-content="Vivamus sagittis lacus vel augue laoreet rutrum faucibus.">
      Popover on right
  </button>
  <button type="button" class="btn btn-secondary" data-container="body" data-toggle="popover" data-
placement="right" data-content="Vivamus sagittis lacus vel augue laoreet rutrum faucibus.">
      Popover on right
  </button>

  <button type="button" class="btn btn-secondary" data-container="body" data-toggle="popover" data-
placement="bottom" data-content="Vivamus
gittis lacus vel augue laoreet rutrum faucibus.">
      Popover on bottom
  </button>

  <button type="button" class="btn btn-secondary" data-container="body" data-toggle="popover" data-
placement="left" data-content="Vivamus sagittis lacus vel augue laoreet rutrum faucibus.">
      Popover on left
  </button>
  <!-- DISMISSIBLE POPOVER -->
  <a tabindex="0" class="btn btn-lg btn-danger popover-dismiss" role="button" data-toggle="popover"
data-trigger="focus" title="Dismissible popover" data-content="And here's some amazing content. It's
very engaging. Right?">Dismissible popover</a>
</div>
```



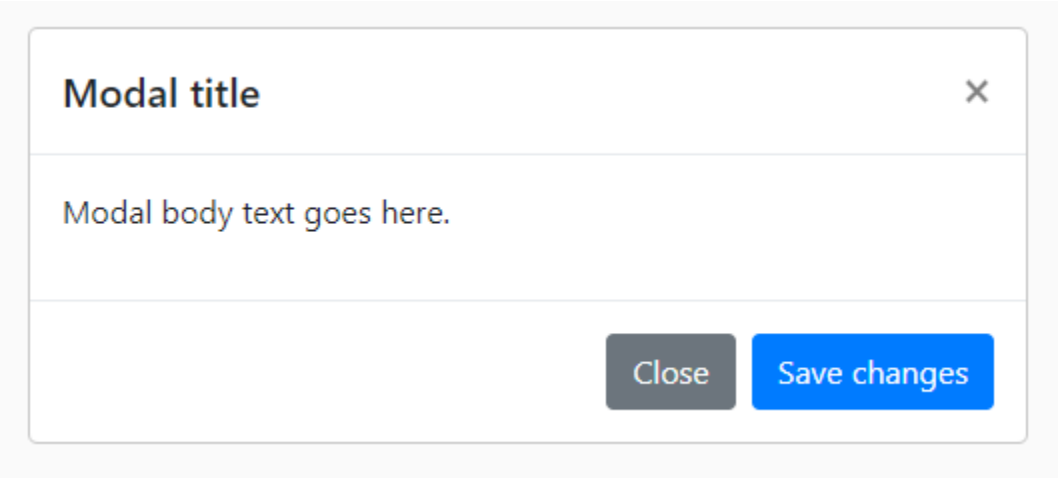
25. Modal Component

This is one of the most used bootstraps JavaScript component used for login and other purposes.

- Modals are built with HTML, CSS, and JavaScript. They're positioned over everything else in the document and remove scroll from the <body> so that modal content scrolls instead.
- Clicking on the modal "backdrop" will automatically close the modal.
- Bootstrap only supports one modal window at a time. Nested modals aren't supported as we believe them to be poor user experiences.

- Modals use position: fixed, which can sometimes be a bit particular about its rendering. Whenever possible, place your modal HTML in a top-level position to avoid potential interference from other elements. You'll likely run into issues when nesting a .modal within another fixed element.

Modal Layout:

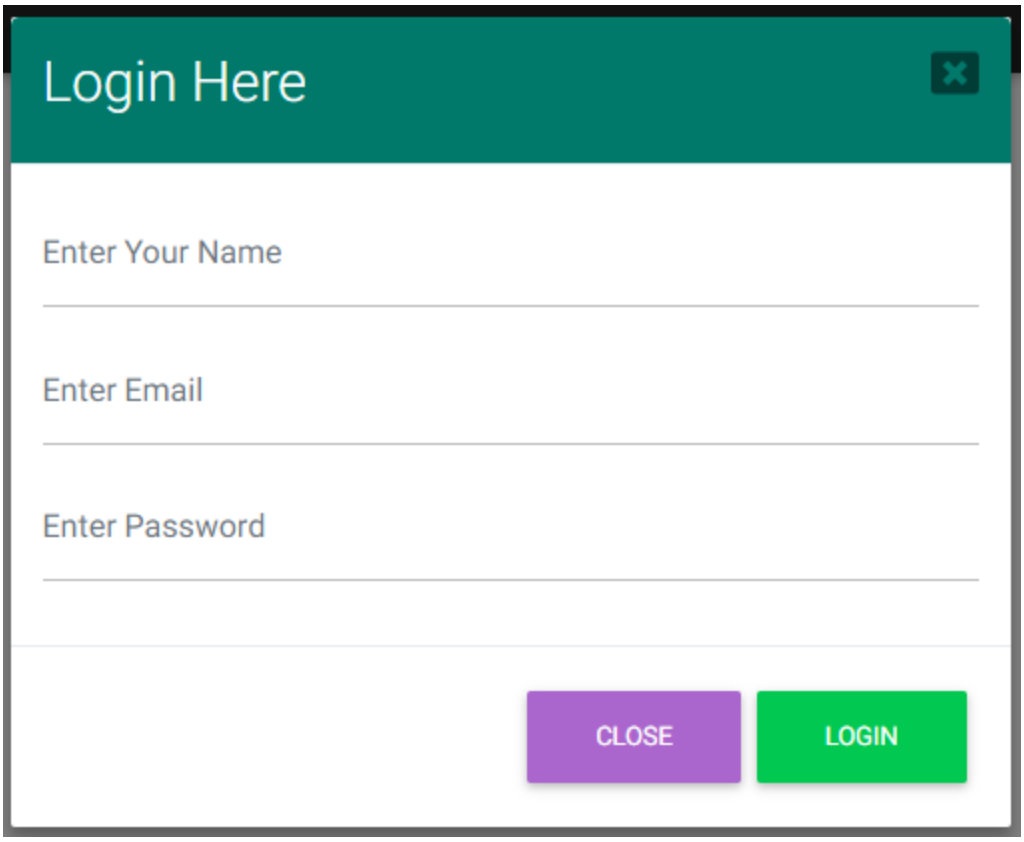



```
<div class="modal" tabindex="-1" role="dialog">
    <div class="modal-dialog" role="document">
        <div class="modal-content">
            <div class="modal-header">
                <h5 class="modal-title">Modal title</h5>
                <button type="button" class="close" data-dismiss="modal" aria-label="Close">
                    <span aria-hidden="true">&times;</span>
                </button>
            </div>
            <div class="modal-body">
                <p>Modal body text goes here.</p>
            </div>
            <div class="modal-footer">
                <button type="button" class="btn btn-primary">Save changes</button>
                <button type="button" class="btn btn-secondary" data-dismiss="modal">Close</button>
            </div>
        </div>
    </div>
</div>
```

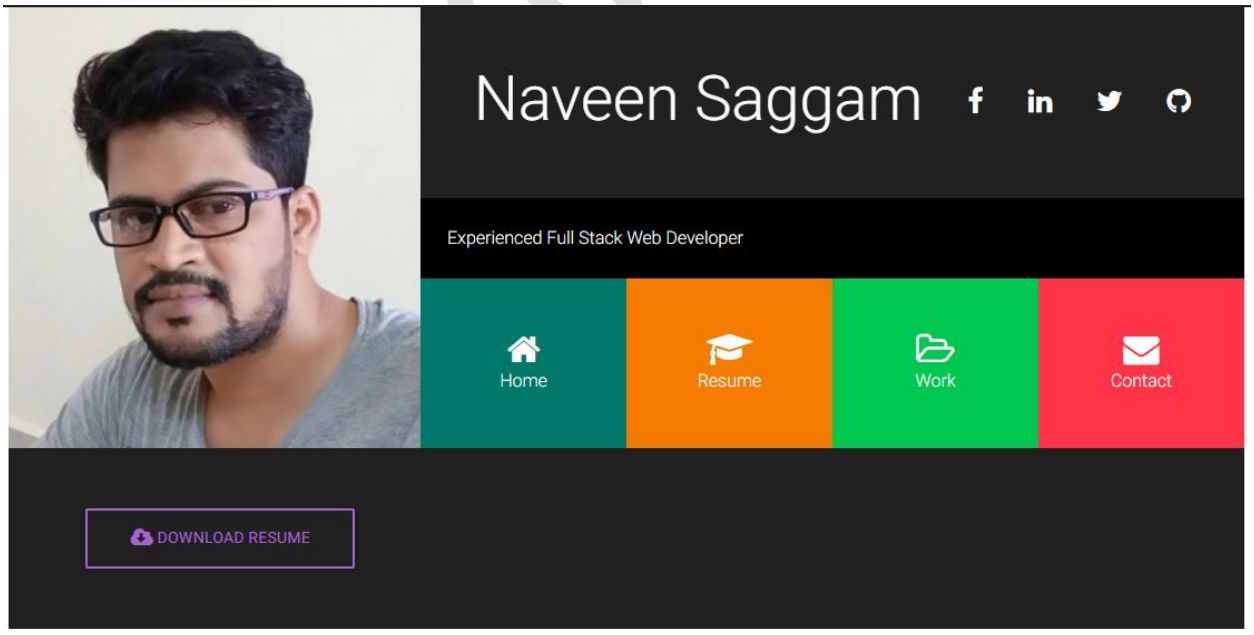
Login Modal

```
<!-- LOGIN MODAL TRIGGER -->
    <a class="btn btn-teal"
       data-toggle="modal" href="#loginModal">Login</a>

    <div id="loginModal" class="modal fade show">
        <div class="modal-dialog">
            <div class="modal-content">
                <form>
                    <div class="modal-header bg-teal text-white">
                        <h3>Login Here</h3>
                        <span class="close" data-dismiss="modal">
                            <i class="fa fa-window-close"></i>
                        </span>
                    </div>
                  <div class="modal-body">
                          <div class="form-group">
                              <input type="text" class="form-control"
                                     placeholder="Enter Your Name">
                          </div>
                          <div class="form-group">
                              <input type="email" class="form-control"
                                     placeholder="Enter Email">
                          </div>
                          <div class="form-group">
                              <input type="password" class="form-control"
                                     placeholder="Enter Password">
                          </div>
                  </div>
                  <div class="modal-footer">
                      <a class="btn btn-secondary" data-dismiss="modal">Close</a>
                      <a class="btn btn-success">Login</a>
                  </div>
              </form>
          </div>
      </div>
  </div>
```

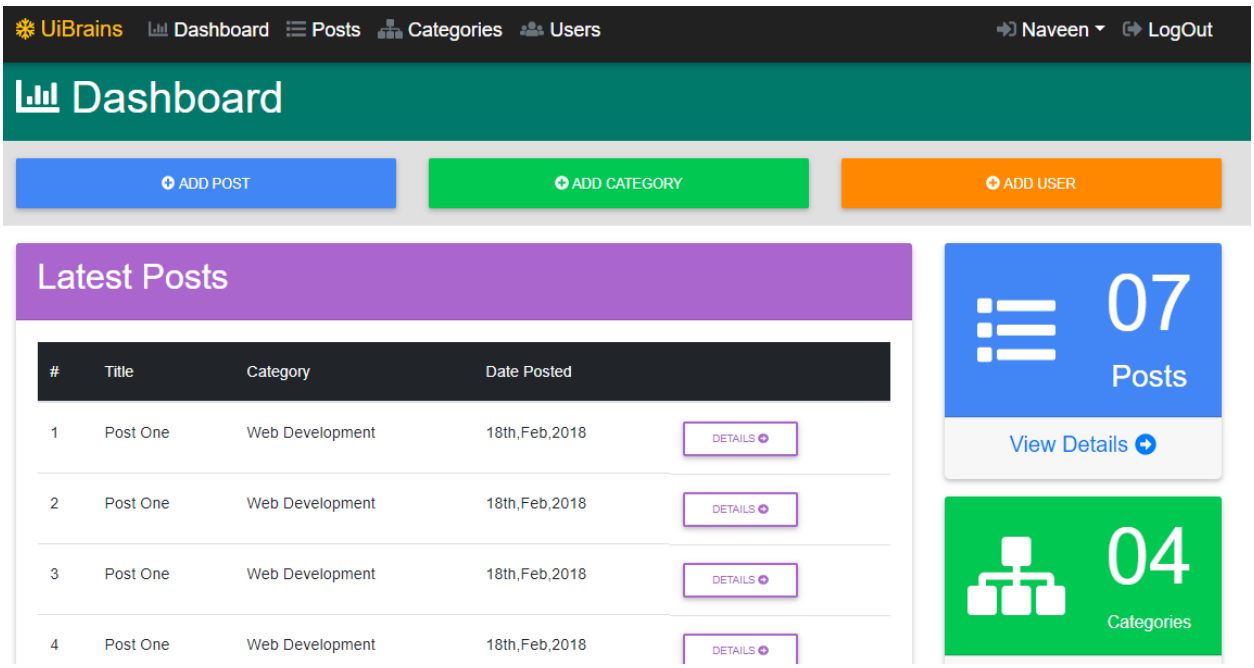
LOGIN



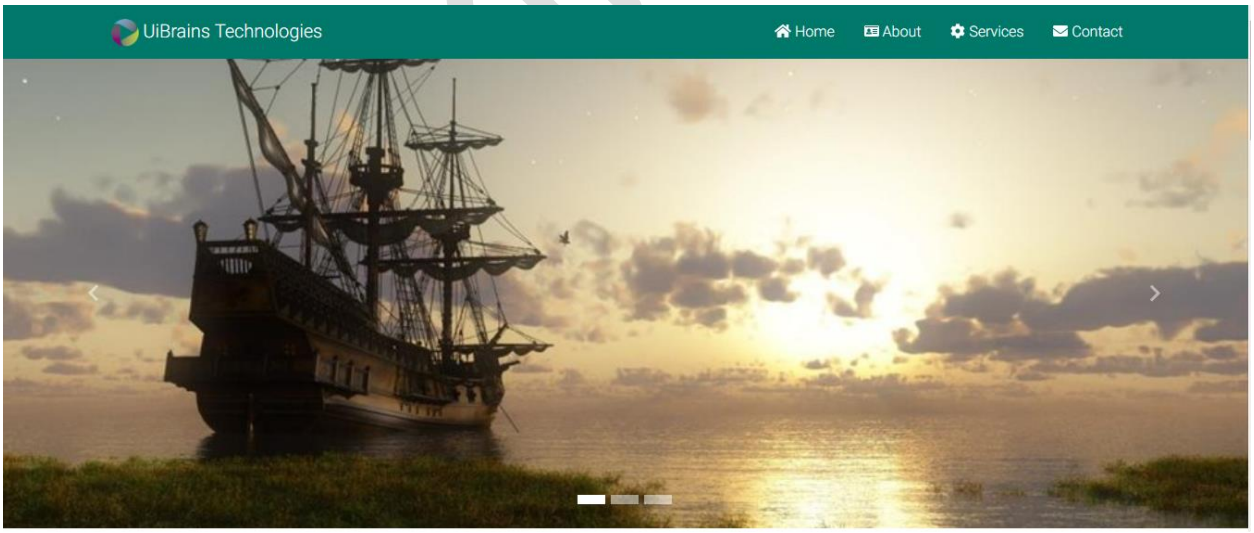
Online Resume Website



UI Dashboard Website



Corporate Website



Best Faculty Members

Lorem ipsum dolor sit amet, consectetur adipisicing elit. Distinctio dolore dolores explicabo omnis provident quae, quod quos sapiente sit temporibus.

Best Coding Skills

Lorem ipsum dolor sit amet, consectetur adipisicing elit. Consequuntur facilis fugit, iusto nam nemo odio repudiandae sint voluptatibus. Alias, distinctio?

Best Course Material

Lorem ipsum dolor sit amet, consectetur adipisicing elit. Animi beatae corporis dolores impedit laborum minus natus officiis quae recusandae tempore?

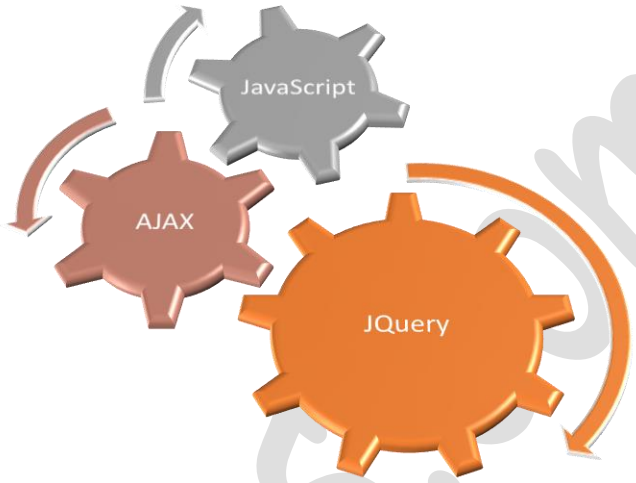
We Provide Best Training for Frontend and Java

Lorem ipsum dolor sit amet, consectetur adipisicing elit. Consequuntur corporis dolorem, eum, facere maiores necessitatibus nesciunt nisi quaerat quis quos recusandae, rem rerum unde. A iste laudantium numquam quasi tenetur!

Please find the courses we offer as below

Lorem ipsum dolor sit amet, consectetur adipisicing elit. Ab, molestiae.

JQuery & AJAX



JQuery

JQuery is a very light weight JavaScript library and the main purpose of JQuery is, it makes much easier to use JavaScript in our webpages.

The biggest feature and main motto of JQuery is “Write less and do more”.

In order to provide a rich dynamic nature for any website we require JavaScript. May be by using JavaScript the code may goes complicated and that can be more simplified using JQuery.

Most of the common tasks of JavaScript may require more number of line of code which are wrapped into more methods in JQuery and they can be called by using single line of code of JQuery.

JQuery simplifies the most complicated things can be done by JavaScript, such as DOM Manipulation and AJAX calls.

Features of JQuery

Using JQuery we can simplify the more complicated things can be done by using JavaScript. The main usage and features of JQuery are as follows,

1. HTML DOM Manipulation
2. CSS Manipulation
3. JavaScript event Handling methods.
4. Effects and Animations.
5. JQuery plugins for complicate and rich UI Components
6. Cross browser compatibility.

Naveen Saggam

HTML DOM Manipulation

JQuery make easy of DOM manipulation. Using we can add or edit or delete DOM Nodes from the DOM Tree and provides a dynamic nature for the content.

Let's understand the JavaScript way and JQuery way of DOM Manipulation.

```
//JavaScript
var text = document.getElementById('textBox').value;

//Jquery
$('#textBox').val();
```

CSS Manipulation

By using the concept called DOM manipulation we can access any HTML and change its css properties using JavaScript dynamically. This even made easy using JQuery.

Let's understand CSS Manipulation using JQuery and JavaScript.

```
//JavaScript
document.getElementById('login-button').style.color='green';

//Jquery
$('#login-button').style.color = 'green';

// JQuery + Bootstrap
$('#form-card-header').removeClass('bg-teal').addClass('bg-success');
```

JavaScript Event Handling methods

JQuery offers an elegant way to capture a wide verity of events, user click on a button or click on a link or any kinds of events. By using JavaScript we may hook up the events and call the JavaScript functions for processing of an event. JQuery provides an easy way of event handing.

Let's understand the event Handling using JavaScript and JQuery is as follows,

```
//JavaScript
document.getElementById('login-button').style.color='green';

//Jquery
$('#login-button').style.color = 'green';

// JQuery + Bootstrap
$('#form-card-header').removeClass('bg-teal').addClass('bg-success');
```

Effects and Animation

JQuery comes with plenty of effects and animations which we can use for our websites.

JQuery effects like hide/show, fade, and slide are more useful to build any rich UI Components. JQuery also comes along with lot of plugins for ready to use features in our websites.

```
$('#success-button').click(function () {
    $('#success-card').fadeToggle('slow');
    var button_value = $(this).attr('value');
    if(button_value === 'HIDE'){
        $(this).attr('value',"SHOW");
    }
    else if(button_value === 'SHOW'){
        $(this).attr('value',"HIDE");
    }
});
```

The above example code we use it for toggle effect for any buttons.

JQuery plugins

A plug-in is a piece of JavaScript code written in a standard JavaScript file. These files provides useful JQuery methods which can be used along with JQuery library methods.

These are ready to use plugins and in order to make use of those plugins, first we needs to add those plugin related JavaScript and CSS files to our HTML file and call the methods specified in the plugin in JQuery way.

There are lot of handpicked plugins available in JQuery website and which allows the user to start using them instead of re-inventing the wheel.

Examples of JQuery plugins are as follows,

JQuery plugins for Form validation, animations, gallery, light box, Menus and Nav, Social Media plugins, Accordions, layouts, sliders and zoom effects.

Cross Browser Compatibility

In the process of UI Development each browser behaves differently for each UI Components. We needs to do some extra work in order to make any website work perfectly in all the browsers.

The jQuery team knows all about cross-browser issues, and they have written this knowledge into the jQuery library. JQuery will run exactly the same in all major browsers.

Installation of JQuery

We do not require any separate installation for JQuery. It needs to add a jquery.js to our application to starts working with it.

We can use the JQuery in two ways such as online version and offline version.

In Offline version we have to download the JQuery.js from JQuery official website and add it to our webpage. Or in Online version we can link JQuery to any CDN like Google.

In this course we will use an Offline version, here we have to download the JQuery.js from official website jquery.com.

There are two versions of JQuery such as Production version and Development version.

Production version

This is a compressed version of JQuery and not human readable.

This version is comparatively less in size than development version. We normally use this for live website or in production environments.

Development Version

This is an un-compressed version and it is human readable.

This version is comparatively more in size than production version. We normally use this for development environments and Testing environments.

We can easily identify the production version and development version with the name of the JavaScript file such as,

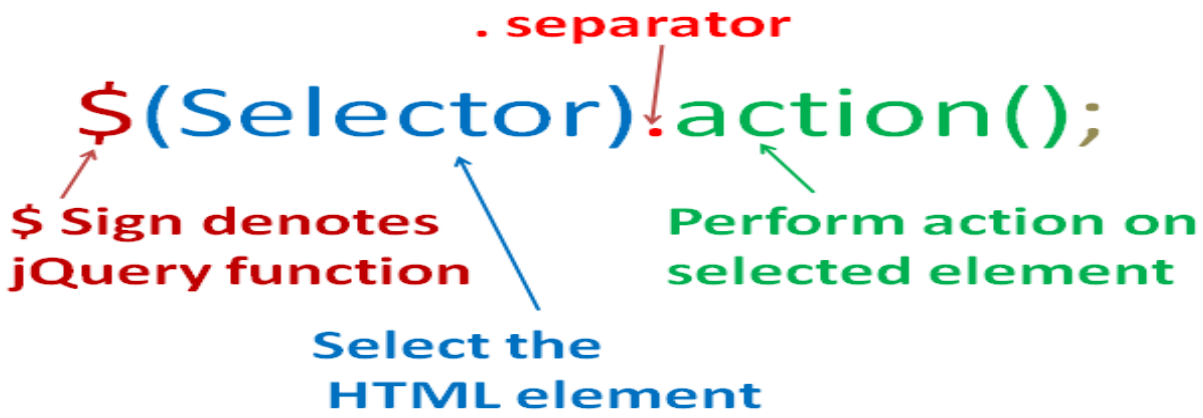
```
Development Version
-------------------
jquery-3.3.1.js

Production Version
------------------
jquery-3.3.1.min.js
```

JQuery Syntax

JQuery is used to select the HTML elements using various selectors and perform some actions on it.

We can identify the JQuery expression using the **$** symbol.

The syntax of JQuery is as follows,



The above diagram explains the complete syntax of JQuery.

Let's see some of the examples of JQuery selectors and their actions as follows,

```
$(this).hide(); // Hides the current element
$('p').hide(); // Hides all the 'p' elements
$('#myCard').toggle('slow'); // Toggles the element
$('.container').hide(); // Hides all the elements with the class 'container'
$('#text').hide(); // Hides the element with id 'text'
```

JQuery Selectors

For JQuery Selectors we can use the same selectors used for CSS.

The type of CSS and JQuery selectors are as follows,

```
Element Selector => $('p')
ID selector => $('#main-nav')
Class selector => $('.container')
Attribute Selector => $("input[type='text']")
Pseudo classes selector => $('tr:even')
```

JQuery Events

Any kind of actions that user can perform on the webpage is called an event. The moment a user is performing an action on the browser is called as an event.

Example:

Selecting a radio button

Focus on an input field

Hover on a button, clicking a button is an event.

Mouse Events	Keyboard Events	Form Events	Document Events
click	keypress	Submit	Load
dbclick	keydown	Change	Resize
mouseenter	keyup	Focus	Scroll
mouseleave			unload

JQuery Basics

We can select the specific HTML element using the JQuery selector. We can apply various actions on that selected HTML Element.

Ex:

```
<p>Lorem ipsum dolor sit amet</p>
```

To hide the above paragraph while performing the click event is as follows,

```
<p onclick="$(this).hide()">Lorem ipsum dolor sit amet</p>
```

We can even select the specific element from outside of the HTML and we can apply the actions on them.

```
<div class="card" id="blue-card">
    <div class="card-body bg-primary text-white text-center">
        <h3 class="card-title">JQuery</h3>
        <p>Lorem ipsum dolor sit amet</p>
    </div>
</div>
```

JQuery

Lorem ipsum dolor sit amet, consectetur adipisicing elit. Asperiores corporis eos labore nemo, optio perspiciatis possimus quisquam veniam! Aliquid dolor eius enim explicabo fugiat ipsa, officia quas quibusdam sed voluptas.

We can hide the above card by clicking on it using JQuery is as follows,

```
<script>
    // Internal JQuery here
    $('#blue-card').click(function () {
        $(this).hide();
    });
</script>
```

We can even apply the toggle effect for the selected HTML Element.

```
<div class="container mt-3">
    <div class="row">
        <div class="col-md-6">
            <input type="button" value="HIDE" class="btn btn-success" id="success-button">
        </div>
        <div class="col-md-6">
            <!-- Empty -->
        </div>
    </div>
```

```
    <div class="row">
        <div class="col-md-8">
            <div class="card" id="success-card">
                <div class="card-body bg-success text-white text-center">
                    <h3>External JQuery</h3>
                    <p>Lorem ipsum dolor sit amet, consectetur adipisicing </p>
                </div>
            </div>
        </div>
        <div class="col-md-4">
            <!-- Empty -->
        </div>
    </div>
</div>
```

```
$('#success-button').click(function() {
    $('#success-card').fadeToggle();
    var btnValue = $(this).attr('value').trim();
    console.log(btnValue.length);
    if(btnValue === 'HIDE'){
        $(this).attr('value','SHOW');
    }
    else{
        $(this).attr('value','HIDE');
    }
});
```

JQuery Event Listeners

The JavaScript events handling is easier in the form of JQuery. Using JavaScript we have to get the element and add an event Listeners and attach a function for them.

JavaScript Way

```
// Get the HTML Element
var buttonElement = document.querySelector('#sumit-button');

// Add an event Listeners
buttonElement.addEventListener('click',hanlingFunction);

// write the function for logic
function handlingFunction() {
    // Write the Logic of DOM Manipulation
}
```

JQuery Way

```
$('#submit-button').click(function () {
    // Write JQuery DOM Logic
});
```

To apply the 'keyup' event for a text box using JavaScript is as follows,

```
// JavaScript way of 'keyup' event

var textBoxElement = document.querySelector('#username');

textBoxElement.addEventListener('keyup',handlingFunction);

function handlingFunction() {
    // Add the required logic
}

// JQuery Way of 'keyup' event
$('#username').keyup(function() {
    // Add Logic
});
```

Description	JavaScript Way	JQuery Way
Select a HTML Element	`document.querySelector('#test');`	`$('#test');`

Add an event Listener	`var element = document.querySelector('#success-button');`
	`element.addEventListener('click',greet);`
	`function greet() {`
	`  // Add logic`
	`}`
		`$('#success-button').click(function() {`
		`  // Add logic`
		`};`

Get input field value	`document.querySelector('#user-text').value;`	`$('#username').val();`
Set text from JS file	`document.querySelector('#greet-msg')` `  .textContent = "Good Morning";`	`$('#greet-msg').text('Good Morning');`
Change CSS Styles	`var element = document.querySelector('#username');` `element.style.borderColor = 'green';`	`$('#username').css("background-color","green");`

AJAX Introduction

Traditionally webpages required reloading to update their content.

For web based emails if users have to reload the page each time to check the new mails is the biggest drawback, as it is slow and requires user input.

For each reload server has to send the completed HTML, CSS, JavaScript code and also the data to the browser.

This approach is highly inefficient and ideally the server should only send the new messages only to the browser instead of the complete page.

By 2003 all the major browsers solved this issue by adopting a new concept called XMLHttpRequest Object.

XMLHttpRequest is a part of a Technology called AJAX.

Using AJAX the data could be passed between the browser and server using XMLHttpRequest API without complete reload of the entire webpage.

Example:

Google Maps uses AJAX to get new map tiles without reload of a webpage.

Gmail gets the new mails without reload the entire web page.

Crickinfo website displays the data without reloading the actual page.

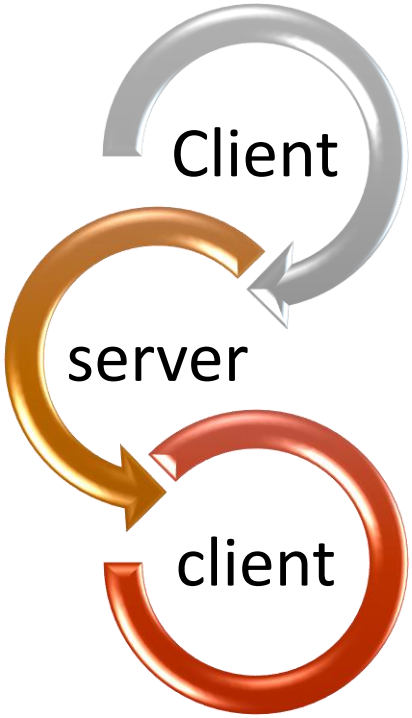
Unfortunately each browser implements the AJAX API differently. But JQuery supports AJAX which works in all the browsers uniformly.

- AJAX Stands for Asynchronous JavaScript and Xml
- AJAX is just a client side programming technique to updates the different parts of the webpage without reloading whole page.
- For This it uses the browsers built-in object called XMLHttpRequest Object.
- This object is responsible to request the data from the webserver for some specific intervals.
- Once we fetch the data from the webserver, we can display the data on the HTML using JavaScript DOM Manipulation or JQuery.
- Using AJAX we can asynchronously get the data from the server. We may get the data in 'xml' format.

Usually the same AJAX technique is used to transfer the data along with the 'xml' , we can even transfer the data in JSON and 'plain Text' format.

How AJAX Works

UiBrains.com



1) An event occurs in the client or the page loads We create an XMLHttpRequest and send to server

2) Server process the Request and prepares the response and send back to the browser

3) Client process the returned data from the server using JavaScript and updates the page contents using HTML DOM Manipulation.

XMLHttpRequest Object

In the whole process of AJAX, this Object places the key role.

This object is responsible to exchange data with a webserver behind the scenes. It means that we can updates the parts of the webpage without reloading the entire webpage.

All the modern browsers have a built in XMLHttpRequest Object.

To create this object we use the below syntax,

```
// Create a AJAX Request
var http = new XMLHttpRequest();
```

```
// Prepare the Request Object to trigger
http.open('GET','db.json',true);

// send to server
http.send();

/*
    Syntax:
    open(method, url, async, user, psw);
    method: the request type GET or POST
    url: the file location
    async: true (asynchronous) or false (synchronous)
    user: optional user name
    psw: optional password
*/
```

The onreadystatechange Property

The readyState property holds the status of the XMLHttpRequest.
The onreadystatechange property defines a function to be executed when the readyState changes.
The status property and the statusText property holds the status of the XMLHttpRequest object.

Property	Description
Onreadystatechange	Defines a function to be called when the readyState property changes
readyState	Holds the status of the XMLHttpRequest. 0: request not initialized 1: server connection established 2: request received 3: processing request 4: request finished and response is ready
Status	200: "OK" 403: "Forbidden" 404: "Page not found"
statusText	Returns the status-text (e.g. "OK" or "Not Found")

Note: The onreadystatechange function is called every time the readyState changes.
When readyState is 4 and status is 200, the response is ready.

Angular JS

Introduction

Angular JS is client side JavaScript framework to build the rich client side applications with MVC architecture and two day binding.

Angular JS is best suitable for Single Page applications.

Angular JS is an open source project and we can use it freely with our applications. This is best suitable to develop complex applications at client side where JavaScript DOM manipulation is almost impossible.

As the applications grows bigger and bigger and more complex logics at the client side, for this we should require client side frameworks. There are many client side frameworks available React JS , Ember , Backbone , Vue JS , Ext JS etc., but still Angular JS is more popular than any other frameworks.

Overall, AngularJS is a framework to build large scale and high performance web application while keeping them as easy-to-maintain

Features of Angular JS

Following are most important core features of AngularJS

Data-binding − It is the automatic synchronization of data between model and view components.

Scope − these are objects that refer to the model. They act as a glue between controller and view.

Controller − these are JavaScript functions that are bound to a particular scope.

Services − AngularJS come with several built-in services for example $https: to make a XMLHttpRequest. These are singleton objects which are instantiated only once in app.

Filters − these select a subset of items from an array and returns a new array.

Directives − Directives are markers on DOM elements (such as elements, attributes, css, and more). These can be used to create custom HTML tags that serve as new, custom widgets. AngularJS has built-in directives (ngBind, ngModel...)

Templates − these are the rendered view with information from the controller and model. These can be a single file (like index.html) or multiple views in one page using "partials".

Routing − It is concept of switching views.

Model View Whatever − MVC is a design pattern for dividing an application into different parts (called Model, View and Controller), each with distinct responsibilities. AngularJS does not implement MVC in the traditional sense, but rather something closer to MVVM (Model-View-ViewModel). The Angular JS team refers it humorously as Model View Whatever.

Deep Linking − Deep linking allows you to encode the state of application in the URL so that it can be bookmarked. The application can then be restored from the URL to the same state.

UiBrains.com

Dependency Injection – AngularJS has a built-in dependency injection subsystem that helps the developer by making the application easier to develop, understand, and test.

Advantages of Angular JS

AngularJS provides capability to create Single Page Application in a very clean and maintainable way.

AngularJS provides data binding capability to HTML thus giving user a rich and responsive experience

AngularJS code is unit testable.

AngularJS uses dependency injection and make use of separation of concerns.

AngularJS provides reusable components.

With AngularJS, developer write less code and get more functionality.

In AngularJS, views are pure html pages, and controllers written in JavaScript do the business processing.

Angular JS way of MVC

If any application logic is becoming more and more complex we have to divide the application into various parts in high level and we should lighten the dependency between them by achieving loose coupling.

In a high level, in order to develop and website we should require the following things, such as

1) Application Design (View)
2) Data for the application (Modal)
3) Logic to change the data. (Controller)

These three parts are club together and called as MVC architecture.

This is one of the successful design pattern for server side, we can also use this client side application if the client side application become more complex.

Model View Controller or MVC as it is popularly called, is a software design pattern for developing web applications. A Model View Controller pattern is made up of the following three parts –

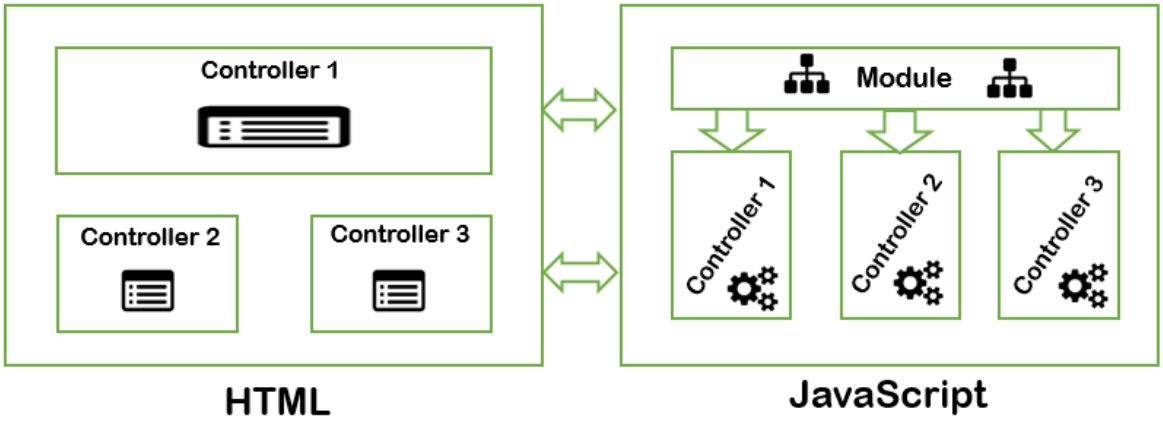
Model – It is the lowest level of the pattern responsible for maintaining data.

View – It is responsible for displaying all or a portion of the data to the user.

Controller – It is a software Code that controls the interactions between the Model and View.

UiBrains.com

Angular JS Architecture



In Angular JS we can design the angular JS application as mentioned above.

In Angular JS we will define a main module for the application and some of other dependent modules, and each module is having various controllers.

Each controller is having its own defined variables and scope objects with their own processing logic. We can connect the angular Module with the HTML using 'ng-app' and we can connect with the Controllers using 'ng-controller' directive.

The 'ng-app' directive is used to starts the angular JS application. Once we define any module with this directive then along with the defined module, all the other dependent modules are also be loaded.

'ng-controller' directive is used to connect the specific controller with parts of HTML Application.

The ng-controller directive attaches a controller class to the view. This is a key aspect of how angular supports the principles behind the Model-View-Controller design pattern.

UiBrains.c

UiBrains.com